
Windows Logo Program

Guidelines for products that work well with the Microsoft® Windows® XP operating systems

“Designed for Microsoft Windows XP” Application Specification

Microsoft®

Version 2.3

January 2, 2002



Portions of this document specify software that is still in development. Some of the information in this documentation may be inaccurate or may not be an accurate representation of the functionality of final documentation or software. Microsoft assumes no responsibility for any damages that might occur directly or indirectly from these inaccuracies.

Microsoft does not make any representation or warranty regarding specifications in this document or any product or item developed based on these specifications. Microsoft disclaims all express and implied warranties, including but not limited to the implied warranties of merchantability, fitness for a particular purpose and freedom from infringement. Without limiting the generality of the foregoing, Microsoft does not make any warranty of any kind that any item developed based on these specifications, or any portion of a specification, will not infringe any copyright, patent, trade secret or other intellectual property right of any person or entity in any country. It is your responsibility to seek licenses for such intellectual property rights where appropriate. Microsoft shall not be liable for any damages arising out of or in connection with the use of these specifications, including liability for lost profit, business interruption, or any other damages whatsoever. Some states do not allow the exclusion or limitation of liability or consequential or incidental damages; the above limitation may not apply to you.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented. This document is for informational purposes only.
MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

Microsoft, MSDN, Windows, the Windows logo, Active Accessibility, DirectX, Direct3D, DirectDraw, DirectSound, DirectInput, IntelliMirror, Win32, Win64, and Windows NT, are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

© 1998 - 2001 Microsoft Corporation. All rights reserved.

Revision History

Date	Changes from Application Specification Version 2.2
January 2, 2002	Added section S5 "Optimized for Enterprises"
January 2, 2002	Revised Section S4.0 "Optimized for .NET"
January 2, 2002	Renamed "Best Practices" sections to "Optimized for"

Date	Changes From Application Specification Version 2.1
September 12, 2001	Reference the Application Compatibility Toolkit for the tools to use for the Logo Program
September 12, 2001	Clarified Section 1.1: 3 mouse buttons <ul style="list-style-type: none">Must not crash when a mouse with more than three buttons is used. You do not need to use all the buttons in your application
September 12, 2001	Dual Proc testing not a requirement (Section 1.1)
September 12, 2001	Do not need to submit documents about your application when submitting your application for the Windows Logo Program
September 12, 2001	Clarification of Section 1.6 Visual Styles <ul style="list-style-type: none">You do not need to use the new themes/visual styles that XP provides, your application just cannot crash/lose functionality when these styles are used
September 12, 2001	.NET Passport Aware (Section S2.7) rolled into Optimized for .NET section
September 12, 2001	Insert DFW – Optimized for .NET Section (Beta requirements). This will be under Section 2, titled S4.0 Optimized for .NET
September 12, 2001	Renamed sections of the Supplemental sections:
September 12, 2001	Rename S4.0 "Future Requirements" to F1.0 "Future Requirements"
October 12	Adding section S5 "Optimized for Enterprises"
October 30, 2001	In section S5, added Heap use, Critical Section use, Handle Check use, and code signing requirements (code signing will not take affect immediately). Moved the no NULL DACL requirement to the future requirements section
April 2, 2002	Replaced section S3 with revised version of the Optimized for Accessibility requirements.

Contents

Welcome.....	5
Resources.....	7
Windows XP Logo Requirements List.....	8
‘Designed For Windows XP - Optimized’ Requirements List	9
Future Guidelines List.....	10
Section 1: Logo Requirements	11
1.0 Windows Fundamentals.....	12
Summary of Windows Fundamental Requirements	12
Windows Fundamentals Requirements	12
2.0 Install/Remove	26
Summary of Installation Requirements	26
Install/Remove Requirements	26
3.0 Data and Settings Management.....	37
Summary of Data Requirements	37
Data and Settings Management Requirements	37
Section 2: ‘Designed For Windows XP - Optimized’ Requirements.....	47
S1.0 Optimized for Games.....	48
Summary of the Optimized for Game Experience Guide	48
Game Experience Guide	48
S2.0 Optimized for Windows XP	54
Summary of the Optimized for Windows XP Guidelines	54
Optimized for Windows XP Guide	54
S3.0 Optimized for Accessibility	60
Summary of Accessibility Guidelines	60
Accessibility Guidelines	61
S4.0 Optimized for .NET	72
Summary of the Optimized for .NET Guidelines	72
Optimized for .NET Guide.....	72
S5.0 Optimized for Enterprises.....	75
Summary of the Optimized for Enterprises Guidelines	75
Optimized for Enterprises Guide.....	76
Section 3: Future Requirements	85
F1.0 Future Requirements.....	85
Summary of Future Requirements	85
Future Requirements.....	85
Glossary	97

Welcome

The *"Designed for Microsoft Windows XP" Application Specification* describes the technical requirements for applications to earn the "Designed for Microsoft Windows XP" logo. Section 1: Logo Requirements in this document provides basic information on the logo program.

These are the top reasons that your customers benefit from an application that meets this compatibility specification:

- The installation and removal of the application are similar to other applications with which the user is familiar.
- Application installation and application execution are less likely to interfere with other applications that the user has already installed.
- The user's first experience with an application is improved because unnecessary reboots are eliminated.
- The application is unlikely to cause the user's computer to fail or function improperly.
- The application makes it easier for family or friends to share a personal computer.
- The application runs in a tightly controlled environment to enable parents to secure the computer but still allow children to run applications, or enable Information Technology professionals to secure the computer properly.
- The application runs correctly on Microsoft® Windows XP, reducing customer frustration and reducing support calls.

"Designed for Microsoft Windows XP" Logo program

The technical requirements defined in *"Designed for Microsoft Windows XP" Application Specification* form the base requirements for applications that are eligible to earn the "Designed for Microsoft Windows XP" logo. This logo tells customers that your application provides a high-quality computing experience on the Windows XP operating system.

An application that is fully compliant with the requirements in this document may carry the "Designed for Microsoft Windows XP" logo after:

- The application has passed self-testing on Windows XP.
- Your company has completed a Logo License Agreement with Microsoft for that specific application.
- Your company has submitted a copy of the application to Microsoft.

For more information about self-testing, what you need to submit to Microsoft, and tools to aid in compliance testing, see

<http://www.microsoft.com/winlogo/software/>, or send email to swlogo@microsoft.com.

"Designed for Microsoft Windows XP" Logo – Optimized Status

If an application meets the core requirements the product has earned the right to carry the Designed for Windows XP logo. If the product meets the core requirements, plus an additional set of requirements (only those requirements which apply to your application), the product will have achieved Designed for Windows XP – Optimized status. Customers will see only the Designed for Windows XP logo, but there are additional marketing benefits for those products that achieve this status.

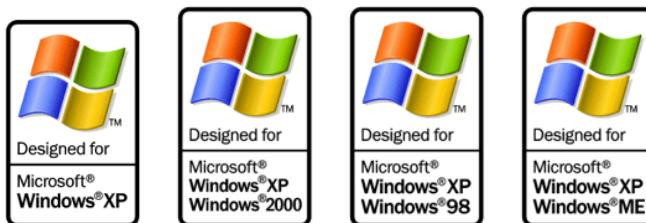
The Designed For Windows XP - Optimized section in this document includes valuable additional guidelines, but it is not required that applications comply with these guidelines to achieve the "Designed for Windows XP" logo. Products that meet the core logo requirements in addition to all of the requirements (which apply to your product) under **one of five** possible Optimized sections will be granted additional marketing benefits from Microsoft (e.g. better placement in the Windows Catalog and other partner opportunities).

Note: Your product can meet the Optimized requirements in one or *more* of the Optimized sections. The more Optimized requirements your product meets, the more marketing benefits you will receive.

Additional Logo Options

Applications may qualify for use of "Designed for Windows XP" logo or any the following logos that include additional Windows versions: Windows 98, Windows ME, Windows NT, or Windows 2000. To use any logo, your application must have been fully tested on Windows XP and passed all of the "Designed for Windows XP" requirements.

If you choose a logo with multiple operating systems listed, your application must also install and perform all of its primary functionality on each of the additional Windows versions. Note that "Windows XP" is always included in any version of the logo.





You may also choose from any of these logos with either a white background or a black background as shown here.



For more information about branding your products with the “Designed for Windows XP” logo, please see Exhibit A to the “Designed for Microsoft Windows” logo license agreement.

Resources

Windows Logo Program Website	http://www.microsoft.com/winlogo
Application Compatibility Toolkit (including Application Verifier)	http://www.microsoft.com/winlogo/downloads/tools.asp
“Designed for Windows XP” for software developers	http://www.microsoft.com/winlogo/software
“Designed for Windows XP” support email	swlogo@microsoft.com
“Certified for Windows” logo program	http://msdn.microsoft.com/certification
Developer information	http://msdn.microsoft.com/windows
Application compatibility Information	http://msdn.microsoft.com/compatibility
Knowledge Base	http://www.microsoft.com/support
Microsoft Platform SDK (Software Developer Kit)	Win32® and Win64™ application programming interfaces (APIs) - http://msdn.microsoft.com/downloads/

Windows XP Logo Requirements List

1.0 Windows Fundamentals

- 1.1 Perform primary functionality and maintain stability
- 1.2 Any kernel-mode drivers that the application installs must pass verification testing on Windows XP
- 1.3 Any device or filter drivers included with the application must pass Windows HCT testing
- 1.4 Perform Windows version checking correctly
- 1.5 Support Fast User Switching and Remote Desktop
- 1.6 Support new visual styles
- 1.7 Support switching between tasks

2.0 Install/Remove

- 2.1 Do not attempt to replace files that are protected by Windows File Protection
- 2.2 Migrate from earlier versions of Windows
- 2.3 Do not overwrite non-proprietary files with older versions
- 2.4 Do not require a reboot inappropriately
- 2.5 Install to Program Files by default
- 2.6 Install any shared files that are not side-by-side to the correct locations
- 2.7 Support Add or Remove Programs properly
- 2.8 Support "All Users" installs
- 2.9 Support Autorun for CDs and DVDs

3.0 Data and Settings Management

- 3.1 Default to the correct location for storing user-created data
- 3.2 Classify and store application data correctly
- 3.3 Deal gracefully with access-denied scenarios
- 3.4 Support running as a Limited User

'Designed For Windows XP - Optimized' Requirements List

S1.0 Optimized for Game Experience

- S1.1 General game playing experience
- S1.2 Do not install or depend on 16-bit files
- S1.3 Do not write to Config.sys, Autoexec.bat, Win.ini, or System.ini files
- S1.4 Provide both manual and automatic install options
- S1.5 Allow installation termination
- S1.6 Running from CD or DVD
- S1.7 Games must be rated
- S1.8 3-D and graphics support
- S1.9 Audio support
- S1.10 Device support
- S1.11 Network support

S2.0 Optimized for Windows XP

- S2.1 Integrate the New Visual Styles
- S2.2 Author high-quality icons
- S2.3 Optimize for Fast User Switching
- S2.4 Write AutoPlay handlers
- S2.5 Follow Control Panel Taxonomy
- S2.6 Enable traversing Network Address Translation

S3.0 Optimized for Accessibility

- S3.1 Support standard system size, color, font, and input settings
- S3.2 Ensure compatibility with the High Contrast option
- S3.3 Enable programmatic access to all user interface elements and text
- S3.4 Provide keyboard access to all features
- S3.5 Expose the location of the keyboard focus
- S3.6 Provide user-selectable equivalents for non-text elements
- S3.7 Do not rely exclusively on sound to convey information
- S3.8 Avoid flashing elements
- S3.9 Create accessible documentation about accessibility features

S4.0 Optimized for .NET

- S4.1 Communication Enhanced Peer-to-Peer Applications
- S4.2 Passport Authenticated Web Service Resource Applications

S5.0 Optimized for Enterprises

- S5.1 Enhanced Reliability
- S5.2 Execute while a virus scanner is running
- S5.3 Degrade gracefully when services are unavailable
- S5.4 Install using a Windows Installer-based package that passes validation testing
- S5.5 Execute appropriately in a multi-lingual environment
- S5.6 Files outside of your application folder have associated file-types
- S5.7 GINAs must support Smart Card login
- S5.8 Support Secure Credential Management
- S5.9 Run in a highly secure configuration
- S5.10 Do not make insecure additions to the secure desktop
- S5.11 Use of network connections must be secure
- S5.12 All executable components must be signed

Future Guidelines List

F1.0 Future Requirements

- F1.1 Install shortcuts correctly
- F1.2 Deliver meaningful user notifications
- F1.3 Support Long Path Names
- F1.4 Operate correctly on high-density displays
- F1.5 Use Windows Installer
- F1.6 Use SHGetFolderPath to determine special folder paths
- F1.7 Implement IPv6 protocol support
- F1.8 Networking applications and protocol support
- F1.9 Create isolated applications using side-by-side assemblies
- F1.10 Do not use NULL DACLs
- F1.11 Do not run as LocalSystem
- F1.12 Use Appropriate Sensitive Data Storage

Section 1: Logo Requirements

These three subsections define the technical requirements for applications to earn the “Designed for Microsoft Windows XP” logo.

1.0 Windows Fundamentals

2.0 Installation Requirements

3.0 Data and Settings Management

1.0 Windows Fundamentals

Summary of Windows Fundamental Requirements

Customer Benefits

Customers can be confident that a compliant product will execute on Microsoft® Windows XP and will not adversely affect the reliability of the operating system.

List of Windows Fundamentals Requirements

1.1	Perform primary functionality and maintain stability
1.2	Any kernel-mode drivers that the application installs must pass verification testing on Windows XP
1.3	Any device or filter drivers included with the application must pass Windows HCT testing
1.4	Perform Windows version checking correctly
1.5	Support Fast User Switching and Remote Desktop
1.6	Support new visual styles
1.7	Support switching between tasks

References

General Functionality and Stability Test Procedure for the “Certified for Microsoft Windows” Logo
<http://msdn.microsoft.com/certification/download.asp>

Using Driver Verifier tool
<http://www.microsoft.com/hwdev/driver/verifier.htm>

Application Compatibility Toolkit
<http://www.microsoft.com/winlogo/downloads/tools.asp>

Windows Fundamentals Requirements

1.1 Perform primary functionality and maintain stability

The application must perform its primary functions without compromising the stability of the operating system or the application.

NOTE The following bullets mention “crash” in several places. For this document, a crash is considered to be an application failure that prevents the user from continuing with the application. A crash may or may not cause data loss. A failure within the application will not be considered a crash if it meets all 3 of the following conditions:

- a) does not cause loss of data,
- b) displays information that would allow a typical user to understand what went wrong and how to avoid the problem in the future

- c) allows the user to continue running the application or close it.

EXAMPLE Performing primary functions without compromising stability:

- If the application creates, edits, and saves multi-page documents, it must not crash or stop responding, and it must not lose the user's data when he or she creates, edits, saves, or opens documents up to the maximum size that you specify.
- If the application displays information about the folders on the user's hard drive, it must not crash or destroy data if the user performs any of the menu functions in the application, such as creating new folders, moving them, or renaming them.

Users must be able to use system features supported by Windows XP with the application. You should test for all of the following:

- Windows XP supports mice with more than three buttons. The application must not crash, stop responding, or lose data when the user presses any button on a supported mouse.
(**Note:** This does not mean your application needs to use more than three buttons. It just cannot crash when a mouse with more than three buttons is used on the system)
- The application must not search for Windows XP system files or temporary folders to be on any particular drive letter by default or assume that it has any maximum or minimum size. Windows XP may be installed on drive letters other than C or D. The C or boot drive can be quite small, under 100 megabytes (MB).
- File and printer names can be long, and Windows supports many characters in these names. The application must not crash or lose data if a user attempts to use long file or printer names.
(**Note:** It is acceptable to block this action and warn the user when he or she attempts this. "S4.0 Future Requirements" explains how to go beyond this to full support of long path and printer names)
- Some editions of Windows can support more than one processor. The application must operate as well on a dual-processor computer as it does on a single-processor computer. (**Note:** This is strongly recommended. Running on a dual processor computer is not a *requirement* for obtaining the DFW logo.)
- If the application uses devices, it must not crash if a device is not installed.

For example, if a user tries to print when a Printer, Fax, or other output device is not installed, the application must degrade gracefully.

- Windows XP does not have an easily user-selectable 256-color mode. If the application requires 256-color mode, the application must switch modes automatically on entry and exit of the application.

Implementation Details - 1.1

For more information about primary functions and stability, see the *“Designed for Microsoft Windows XP” Application Test Framework*, located at

<http://www.microsoft.com/winlogo/downloads/software.asp>.

Test Cases - 1.1

As defined in *“Designed for Microsoft Windows XP” Application Test Framework*:

- TC1.1.1 Does application perform its primary functions and maintain stability during functionality testing?
- TC1.1.2 Does application remain stable when a mouse with more than three buttons was used?
- TC1.1.3 Does application use the user’s temporary folder for temporary files?
- TC1.1.3.1 Does application store its temporary files only in the user’s temporary folder during installation?
- TC1.1.3.2 Does application store its temporary files only in the user’s temporary folder during functionality testing?
- TC1.1.4 Does application not crash or lose data when presented with long path, file and printer names?
- TC1.1.4.1 Does application maintain stability when a file is saved by drilling down through the “User1 LFNPath1” path in User1’s “My Documents” folder?
- TC1.1.4.2 Does application maintain stability when a file is saved by entering the full “User1 LFNPath2” path?
- TC1.1.4.3 Does application maintain stability when a file is saved using a long file name?
- TC1.1.4.4 Does application maintain stability when a file is opened by drilling down through the “User1 LFNPath1” path in User1’s “My Documents” folder?
- TC1.1.4.5 Does application maintain stability when a file is opened by entering the full “User1 LFNPath2” path?
- TC1.1.4.6 Does application maintain stability when a file is opened using a long file name?
- TC1.1.4.7 Does application maintain stability when printing to a printer with a long name?
- TC1.1.5 Does application perform primary functionality and maintain stability on a dual-processor computer?
- TC1.1.6 Does application not crash when devices it uses are not installed?
- TC1.1.6.1 Does application maintain stability when printing if no printer is installed?
- TC1.1.6.2 Does application maintain stability when attempting to use devices that are not installed?
- TC1.1.7 Does application switch the system’s display mode back to the previous color mode, if application automatically changes to 256-color mode when it runs?

1.2 Any kernel-mode drivers that the application installs must pass verification testing on Windows XP

Poorly written kernel-mode drivers have the potential to crash the system. Therefore, it is critical that any application that includes kernel-mode drivers, such as backup, copy protection and compact disc (CD) burning products, be thoroughly tested to minimize this risk.

If the application includes any kernel-mode drivers, each of these drivers must pass validation testing under the Windows Driver Verifier Manager tool (Verifier.exe). That is, Driver Verifier must not report any stop error messages, or otherwise cause system instability, while exercising the system and your application with Driver Verifier installed on your kernel-mode components.

You are responsible for ensuring that all components provided with your application meet this requirement.

Implementation Details - 1.2

Driver Verifier is located in the \system32 directory on systems running Windows XP. For more information about using the Driver Verifier Tool and diagnosing driver problems, see <http://www.microsoft.com/hwdev/driver/verifier.htm>.

Test Cases - 1.2

As defined in “*Designed for Microsoft Windows XP*” Application Test Framework:

- TC1.2.1 Do all related kernel-mode drivers pass testing as Windows XP loaded them?
- TC1.2.2 Do all related kernel-mode drivers pass functionality testing with standard kernel testing enabled?
- TC1.2.3 Do all related kernel-mode drivers pass low-resources simulation testing?

1.3 Any device or filter drivers included with the application must pass Windows HCT testing

If the product includes drivers for hardware devices and filter drivers, these drivers must pass the related tests provided in Windows Hardware Compatibility Test (HCT) 10.0 or later.

For certain categories of drivers, Windows XP will present a warning to end users if they attempt to install a driver that does not have a digital signature from Microsoft. For any driver categories that require a digital signature, the component must be digitally signed by Microsoft.

You are responsible for ensuring that all components provided with your application meet this requirement.

Implementation Details - 1.3

For more information about the HCT and digital signatures, see <http://www.microsoft.com/hwtest>.

Test Cases - 1.3

As defined in *"Designed for Microsoft Windows XP" Application Test Framework:*

- TC1.3.1 Are proofs of WHQL testing attached to the submission for all required drivers?
- TC1.3.2 Do no warnings appear about unsigned drivers during testing?

1.4 Perform Windows version checking correctly

The application must verify that the operating system meets the minimum version requirements for the application. The application must also install and run on all later versions of Windows.

EXAMPLE If the application requires Microsoft Windows NT® 4.0 with Service Pack 3 (SP3), version checking should allow installation on Windows NT, Major Version 4, Minor Version 0, SP3. It must also install on all operating system versions later than this number, such as Windows NT 4.0 Service Pack 4, Windows 2000, Windows XP, Windows XP with any subsequent Service Packs, and so on.

WHEN DOES THIS APPLY?

In certain cases, it is acceptable to block installation of the application on later versions of Windows. If you choose to do this, you must display a *clear* message to the user when blocking installation or execution that the application is not designed for the later Windows version.

An example in which blocking installation might be appropriate is low-level disk utilities. In this case, running such an application on a Windows version for which the product was not tested for could potentially result in lost user data if there were changes in the file system on a later version of Windows.

Implementation Details - 1.4

In general, use the **GetVersionEx** API to determine the Windows version.

Code sample to verify the Windows version requirements

This code runs on all 32-bit Windows platforms.

```
BOOL bIsWindowsVersionOK(      DWORD dwWin9xMajor,  DWORD dwWin9xMinor,
                               DWORD dwWinNTMajor,  DWORD dwWinNTMinor,  WORD
                               wWinNTSPMajor )
{
    OSVERSIONINFO          osvi;
    // Initialize the OSVERSIONINFO structure.
    ZeroMemory( &osvi, sizeof( osvi ) );
    osvi.dwOSVersionInfoSize = sizeof( osvi );
    GetVersionEx( &osvi );  // Assume this function succeeds.
```

```

// Split code paths for NT and Win9x
if( osvi.dwPlatformId == VER_PLATFORM_WIN32_WINDOWS )
{
    // Check the major version.
    if( osvi.dwMajorVersion > dwWin9xMajor )
        return TRUE;
    else if( osvi.dwMajorVersion == dwWin9xMajor )
    {
        // Check the minor version.
        if( osvi.dwMinorVersion >= dwWin9xMinor )
            return TRUE;
    }
}
else if( osvi.dwPlatformId == VER_PLATFORM_WIN32_NT )
{
    // Check the major version.
    if( osvi.dwMajorVersion > dwWinNTMajor )
        return TRUE;
    else if( osvi.dwMajorVersion == dwWinNTMajor )
    {
        // Check the minor version.
        if( osvi.dwMinorVersion > dwWinNTMinor )
            return TRUE;
        else if( osvi.dwMinorVersion == dwWinNTMinor )
        {
            // Check the service pack.
            DWORD dwServicePack = 0;

            if ( _tcsclen( osvi.szCSDVersion ) > 0 )
            {
                _stscanf( osvi.szCSDVersion,
                    _T("Service Pack %d"),
                    &dwServicePack );
                return ( dwServicePack >= wWinNTSPMajor );
            }
        }
    }
}
return FALSE;
}

```

VerifyVersionInfo, a Windows XP API, compares a set of operating system version requirements with the corresponding values for the currently running version of Windows. There are many options and ways to use **VerifyVersionInfo**. However, to check whether the Windows version is new enough, call the function using the flags for checking the major version, minor version, and service pack flags.

VerifyVersionInfo is available only on Windows 2000 and later versions of Windows.

EXAMPLE Specify that “My application needs Windows 2000, with SP 1, or later” and then run **VerifyVersionInfo** to find out, “Is this OS that I’m running up to that standard?” **VerifyVersionInfo** returns either a true or a false.

Example of flags: major version, minor version, and service pack

```
VerifyVersionInfo(&osvi,
    VER_MAJORVERSION |
    VER_MINORVERSION |
    VER_SERVICEPACKMAJOR,
    dwlConditionMask);
```

Example of versioning code

```
BOOL bIsWindowsVersionOK(DWORD dwMajor, DWORD dwMinor, DWORD dwSPMajor )
{
    OSVERSIONINFOEX osvi;
    ZeroMemory(&osvi, sizeof(OSVERSIONINFOEX));
    osvi.dwOSVersionInfoSize = sizeof(OSVERSIONINFOEX);
    osvi.dwMajorVersion = dwMajor;
    osvi.dwMinorVersion = dwMinor;
    osvi.wServicePackMajor = dwSPMajor;
    // Set up the condition mask.
    VER_SET_CONDITION( dwlConditionMask, VER_MAJORVERSION,
        VER_GREATER_EQUAL );
    VER_SET_CONDITION( dwlConditionMask, VER_MINORVERSION,
        VER_GREATER_EQUAL );
    VER_SET_CONDITION( dwlConditionMask, VER_SERVICEPACKMAJOR,
        VER_GREATER_EQUAL );
    // Perform the test.
    return VerifyVersionInfo(&osvi,
        VER_MAJORVERSION | VER_MINORVERSION |
        VER_SERVICEPACKMAJOR,
        dwlConditionMask);
}
```

Test Cases - 1.4

As defined in *“Designed for Microsoft Windows XP” Application Test Framework*:

- TC1.4.1 Does application install correctly under current and future versions of Windows?
- TC1.4.2 Does application perform all functionality tests correctly under current and future versions of Windows?

1.5 **Support Fast User Switching and Remote Desktop**

In Windows XP, the Fast User Switching feature allows multiple users sharing the same computer to have individual profiles and to swap their current work spaces without logging off. The application must not crash or lose data or settings when customers use Fast User Switching.

For example, if the first user has an editor application open and a subsequent user launches the same editor application, the first instance of the application must not shut down and must not lose any of the first user’s edits.

Remote Desktop lets you create a virtual session on your desktop computer using Microsoft's Remote Desktop Protocol (RDP). With Remote Desktop, you can access all of the data and applications stored on your desktop computer from virtually any network connection, including a dial-up or VPN connection. In general, Remote Desktop will be supported if your application supports Fast User Switching. However, see note on Graphical Identification and Authentication dynamic link libraries (GINA's) below.

By meeting the requirements for Data and Settings Management, your application is unlikely to have problems with Fast User Switching, except for possible conflicts with shared resources such as CD drives, printers, modems, and sound cards.

If additional instances of the application run by separate users can result in failure of primary functionality, you must do one of the following:

- Detect that it is already running under a separate user account and block the specific potentially problematic features, or
- Detect that it is already running and block all features of the application when launching subsequent instances of the application. You need to make the user who launches the second (or later) instance of the application aware that they will not be able to use the product. Please use meaningful messages.

When blocking any feature to prevent failure under Fast User Switching and Remote Desktop, the application must inform the user why it did so.

WHEN DOES THIS APPLY?

This clause applies somewhat differently for applications installing a Graphical Identification and Authentication dynamic link library (GINA). A correctly written GINA performs all identification and authentication user interactions, and by design all GINA's replace the Windows XP welcome screen and disable Fast User Switching. Applications that install a GINA must satisfy both of the following:

- Upon installation to a computer that is not on a domain, the setup must warn the user that continuing the installation will disable Fast User Switching and replace the Windows welcome screen.
- The GINA must work correctly with Remote Desktop

Implementation Details – 1.5

A common way to detect another running instance of the application is to use **FindWindow** or **FindWindowEx** to search for a window that the application opens.

Note that **FindWindow** and **FindWindowEx** will not find a window that is open in another user's session. Therefore, if the application

shares some global resource in an unsafe way, you will need to use a different technique to detect another instance of the application.

Another common way to detect another instance of the application is to create a mutex or semaphore when the application is opened and close it when the application exits. Unless you are using a global mutex, this technique will not find another instance of the application, if the other instance is running in another user’s session. This is because the local object namespace is separated for each desktop, allowing a unique list of mutexes and semaphores for each user’s session.

To prevent an instance of the application from running when the previous instance is in another user’s session, you need to give your mutex or semaphore a global namespace name. You do this by prefixing the object’s name with **Global**. This allows you to detect instances of the application in other users’ sessions.

Note that versions of Windows other than Windows 2000 and Windows XP do *not* support global namespace names and will not operate correctly if you use a backslash (\) in the object name.

Sample code - DoIExist:

```
HANDLE g_hMutexAppIsRunning = NULL;

BOOL DoIExist(LPCTSTR szMutexName, BOOL bGlobally)
{
    BOOL bOSSupportsGlobalMutexes = FALSE;
    OSVERSIONINFO OsVersionInfo;
    TCHAR szGlobalPrefix[] = TEXT("Global\\");
    TCHAR szFinalMutexName[256]; //Assume length mutex name not >248

    OsVersionInfo.dwOSVersionInfoSize = sizeof(OsVersionInfo);

    // Check for Windows 2000 (NT 5.0) or higher
    // Note: bOSSupportsGlobalMutexes is defined in requirement 1.4
    bOSSupportsGlobalMutexes = bIsWindowsVersionOK( 9, 0, 5, 0, 0 );

    if (bGlobally && bOSSupportsGlobalMutexes)
    {
        // Prepend "Global\" to mutex name
        _tcscpy(szFinalMutexName, szGlobalPrefix);
        _tcscat(szFinalMutexName, szMutexName);
    }
    else
    {
        _tcscpy(szFinalMutexName, szMutexName);
    }

    // Create or open a named mutex.
    g_hMutexAppIsRunning = CreateMutex( NULL, FALSE,
                                         szFinalMutexName);

    // Close handle, and return NULL if existing semaphore opened
    if (g_hMutexAppIsRunning != NULL &&
        GetLastError == ERROR_ALREADY_EXISTS)
    {
        CloseHandle(g_hMutexAppIsRunning);
        g_hMutexAppIsRunning = NULL;
        return TRUE;
    }

    // If new semaphore was created, return FALSE.
    return (g_hMutexAppIsRunning == NULL);
}
```

Using DoIExist with a Global mutex:

```
if ( DoIExist(TEXT("Savvy is Running"), TRUE) == TRUE )
{
    HWND hWndMe;
    hWndMe = FindWindow(szWindowClass, szTitle);
    if (hWndMe)
    {
        if (IsIconic(hWndMe))
        {
            ShowWindow(hWndMe, SW_RESTORE);
        }
        SetForegroundWindow(hWndMe);
    }
    else
        MessageBox(NULL,
                   TEXT("Another User is using this Application"),
                   szTitle, MB_OK);
    return FALSE;
}
```

Note that when your program exits, you must free the global mutex:

On exit, free the Global mutex

```
if (g_hMutexAppIsRunning != NULL)
{
    CloseHandle(g_hMutexAppIsRunning);
    g_hMutexAppIsRunning = NULL;
}
```

The previous example also shows what to do if your application is preventing a subsequent instance of the application. In previous versions of Windows, you could safely assume that **FindWindow** and **FindWindowEx** would be able to locate the other instance of the application. With Windows XP and Fast User Switching, this may not be as safe.

When running on Windows XP or later, your application should check to ensure that **FindWindow** and **FindWindowEx** have succeeded. If **FindWindow** or **FindWindowEx** has found the other instance, switching to that instance is the correct action.

However, if using a global mutex detects a previous instance and **FindWindow** or **FindWindowEx** is not successful, be sure to notify the user that the application is not starting because another user is already running it. Otherwise, the application will exit without giving the user any indication of why it does not work.

Replacement GINA

For Applications that install a replacement GINA, the replacement GINA must adhere to current Winlogon APIs. In Windows XP there are some scenarios where not using the current Winlogon APIs breaks interface versioning & transparent cross session credential

transfers. These breaks can cause failures of major Windows XP features such as Remote Desktop. By design all replacement GINA's will disable the Windows XP welcome screen and Fast User Switching.

The new Winlogon APIs (Wlx APIs) that must be utilized by the replacement GINAs are WlxGetConsoleSwitchCredentials & WlxQueryConsoleSwitchCredentials. The APIs will export a new entry point to read logged on user credentials and versioning information.

WlxGetConsoleSwitchCredentials

The WlxGetConsoleSwitchCredentials is called by Winlogon to read the currently logged on user's credentials to transparently transfer them to a target session.

```
BOOL WINAPI WlxGetConsoleSwitchCredentials (
    PVOID pWlxContext,
    PVOID pInfo
);
```

Parameters
pWlxContext

[in] Pointer to a GINA specific context.

pInfo

[out] Pointer to a WLX_CONSOLESWITCH_CREDENTIALS_INFO_V1_0 to return GINA relevant information.

Return Values

Returns TRUE on success and FALSE on failure.

WlxQueryConsoleSwitchCredentials

The WlxQueryConsoleSwitchCredentials callback function is called by GINA to read the credentials transferred from the Winlogon of the temporary session to the Winlogon of the destination session.

```
WlxQueryConsoleSwitchCredentials(
    PWLX_CONSOLESWITCH_CREDENTIALS_INFO_V1_0 pCred
);
```

Parameters
pCred

[out] Pointer to a WLX_CONSOLESWITCH_CREDENTIALS_INFO_V1_0 structure to be filled with credentials information.

Return Value:

Returns TRUE if credentials were transferred and FALSE if the transfer failed.

Remarks

In order to access this function, the GINA DLL must use the WLX_DISPATCH_VERSION_1_4 structure

```
WLX_DISPATCH_VERSION_1_4 WlxDispatchTable = {
```

```
    WlxUseCtrlAltDel,
    WlxSetContextPointer,
    WlxSasNotify,
```

```
WlxSetTimeout,
WlxAssignShellProtection,
WlxMessageBox,
WlxDialogBox,
WlxDialogBoxParam,
WlxDialogBoxIndirect,
WlxDialogBoxIndirectParam,
WlxSwitchDesktopToUser,
WlxSwitchDesktopToWinlogon,
WlxChangePasswordNotify,
WlxGetSourceDesktop,
WlxSetReturnDesktop,
WlxCreateUserDesktop,
WlxChangePasswordNotifyEx,
WlxCloseUserDesktop,
WlxSetOption,
WlxGetOption,
WlxWin31Migrate,
WlxQueryClientCredentials,
WlxQueryInetConnectorCredentials,
WlxDisconnect,
WlxQueryTerminalServicesData,
WlxQueryConsoleSwitchCredentials,
WlxQueryTsLogonCredentials
```

```
};
```

Test Cases - 1.5

As defined in *“Designed for Microsoft Windows XP” Application Test Framework*:

- TC1.5.1 Does application properly support Fast User Switching?
- TC1.5.2 Does application properly support Remote Desktop?
- TC1.5.3 If the application installs a replacement GINA, does the GINA properly support Remote Desktop?

1.6 Support new visual styles

If the application loses functionality or usability when a user selects one of the new visual styles, you must disable the style for the application. (Note: You do not need to use the new themes/visual styles that XP provides, but the application must not crash/lose functionality when these styles are used)

Applications that are full-screen graphics typically do not need to do anything to meet this requirement, because changing the visual styles in Windows would be unlikely to have an adverse affect on the functionality or usability of the application.

Implementation Details - 1.6

To disable the new visual style for a top-level window, consider the following:

- As long as a window has a non-NULL region applied to it (**SetWindowRgn**), the Theme Manager assumes that this is a specialized window and the window will not use visual styles. A child window associated with a non-visual style top-level window may still apply visual styles even though the parent window does not.
- If you want to disable the use of visual styles for any top-level window in the application, call **SetThemeAppProperties** and do not pass the STAP_ALLOW_NONCLIENT flag.
- If an application does not call **SetThemeAppProperties**, the assumed flag values are STAP_ALLOW_NONCLIENT | STAP_ALLOW_CONTROLS | STAP_ALLOW_WEBCONTENT. The assumed values cause the non-client area, the controls, and Web content to have a visual style applied.

Test Cases - 1.6

As defined in “*Designed for Microsoft Windows XP*” Application Test Framework:

TC1.6 Does application pass all functionality tests with a Windows XP theme applied?

1.7 Support switching between tasks

The application must not block ALT+TAB, CTRL+ALT+DEL, CTRL+SHIFT+ESC and other task-switching scenarios. The application must *not* try to defeat these mechanisms in any way.

Test Cases - 1.7

As defined in “*Designed for Microsoft Windows XP*” Application Test Framework:

TC1.7.1 Does application display normally and not lose data when focus is switched among other applications with Alt+Tab?

TC1.7.2 Does application display normally and not lose data when Windows logo key and the taskbar are used to switch among applications?

TC1.7.3 Does the Windows Security dialog box or the Task Manager display normally and can application be cancelled or closed without losing data?

2.0 Install/Remove

Summary of Installation Requirements

Customer Benefits

Customers can be confident that your product will install onto Microsoft® Windows XP without degrading the operating system or other applications.

Installation Requirements List

- 2.1 Do not attempt to replace files that are protected by Windows File Protection
- 2.2 Migrate from earlier versions of Windows
- 2.3 Do not overwrite non-proprietary files with older versions
- 2.4 Do not require a reboot inappropriately
- 2.5 Install to Program Files by default
- 2.6 Install any shared files that are not side-by-side to the correct locations
- 2.7 Support Add or Remove Programs properly
- 2.8 Support "All Users" installs
- 2.9 Support Autorun for CDs and DVDs

Install/Remove Requirements

2.1 Do not attempt to replace files that are protected by Windows File Protection

The application must not attempt to replace any files that are protected by Windows File Protection (WFP). To ensure that the application does not invoke WFP, it should call **SfcIsFileProtected** when installing any file that it did not create. The Windows Installer service does this automatically.

Protected files include the following files that ship on the Windows XP product CD:

- Most .SYS, .DLL, .EXE and .OCX files.
- The following fonts: Micross.ttf, Tahoma.ttf, Tahomaabd.ttf, Dosapp.fon, Fixedsys.fon, Modern.fon, Script.fon, and Vgaoem.fon.

NOTE Some redistributable files, such as specific versions of Microsoft Foundation Classes (MFC) DLLs, are installed by Windows XP and are protected by WFP.

Protected files form the core of the operating system and it is essential for system stability that the proper versions be maintained. These files can only be updated through service packs, operating system upgrades, Quick Fix Engineering (QFE) hot-fixes, and Windows Update. Applications cannot replace them, and attempting to replace these files by any means other than those listed above will

result in the files being restored by the Windows File Protection feature (see the subsection **About Windows File Protection**, below).

If the application requires newer versions of these components, it must update these components by using a Microsoft Service Pack that installs the required versions.

EXAMPLE When Microsoft publishes an update to DirectX, it will be provided in a package (either a Windows service pack or its own service pack). An application including the updated DirectX must use the package install and not attempt to directly install files from the package. Installing individual files is not allowed under the Logo Program requirements; in addition, Windows File Protection would prevent it and the user experience would be poor.

About Windows File Protection

Windows File Protection is a feature of Windows XP that prevents the unauthorized replacement of essential system files. WFP runs as a background process on Windows XP and monitors the files listed earlier in this section. When WFP detects that a protected file has been changed, it restores the original.

Do not prompt the user to update or delete any Windows File Protected components.

NOTE Attempting to install components that are under Windows File Protection but have not yet been installed on the system will cause Windows File Protection to install the components. This is correct behavior.

Implementation Details - 2.1

The following code shows how to check whether a file (in this example, “ntdll.dll”) is protected by WFP. Note that

SfcIsFileProtected is Unicode-only and requires a fully qualified path.

```
SHGetFolderPath(NULL,CSIDL_SYSTEM, NULL, 0, szSystemDir);
PathAppend(szSystemDir,"ntdll.dll");
MultiByteToWideChar(CP_ACP, 0, szSystemDir, -1, wzFileName, 265);

if ( SfcIsFileProtected(wzFileName) )
    MessageBox(hWnd,szProtected,szSystemDir,MB_OK);
else
    MessageBox(hWnd,szNotProtected,szSystemDir,MB_OK);
```

Test Cases - 2.1

As defined in “*Designed for Microsoft Windows XP*” Application Test Framework:

TC2.1	Does the installation finish without any Windows File Protection messages appearing?
-------	--

2.2 Migrate from earlier versions of Windows

The application must continue to function correctly when the operating system is upgraded to Windows XP from Windows 98,

Windows ME, Windows NT, or Windows 2000. It is strongly recommended that the application do this without requiring un-installation and reinstallation.

If your application requires Windows version specific code (kernel mode drivers, VxDs, etc.), you must ensure that a solution is available to the user for download or retrieval when they upgrade to a later version of Windows.

- The application must not crash or lose data if run after the system is upgraded to a new version.
- The solution must be freely available.
- The solution must not require any technical actions. It must be easy to use and targeted for end-users. For example, it must not require the user to modify registry entries or change INI settings

To achieve this, the application may need to be able to react to operating system changes dynamically at runtime. Many applications detect the Windows version during installation and decide what to install based on that. This usually means that the installed application is operating system-specific, and when Windows is upgraded to a newer version it will not function properly.

WHEN DOES THIS APPLY?

Applications only need to migrate from versions of Windows that they support. For example, if the application does not install onto Windows 98 or Windows ME, it does not need to migrate from those versions.

Implementation Details – 2.2

For example, consider a case where a music player requires an operating system-specific sound library, which is either Sndlib9x.dll or Sndlbt.dll (representing Window 9x and Windows NT, respectively).

Incorrect:

1. Installer users **GetVersionEx** and detects platform as VER_PLATFORM_WIN32_WINDOWS. It installs Sndlib9x.dll, because that operates properly on the Windows 9x platform.
2. User migrates to Windows XP.
3. Upon launch of the music player, a crash occurs because Sndlib9x.dll doesn't run on NT-based operating systems.

Recommended solution:

1. Installer copies both Sndlib9x.dll and Sndlbt.dll to the program directory during installation.
2. User migrates to Windows XP.
3. Upon launch, the music player detects the platform and decides which library to use. In this case, it will load Sndlbt.dll because that works properly on NT-based operating systems.

The latter example will allow the application to migrate without reinstallation.

Test Cases - 2.2

As defined in “*Designed for Microsoft Windows XP*” Application Test Framework:

- TC2.2.1 Does application successfully migrate from Windows 98 to Windows XP Home Edition?
- TC2.2.2 Does application successfully migrate from Windows Me to Windows XP Home Edition?
- TC2.2.3 Does application successfully migrate from Windows 98 to Windows XP Professional?
- TC2.2.4 Does application successfully migrate from Windows Me to Windows XP Professional?
- TC2.2.5 Does application successfully migrate from Windows NT 4.0 Workstation to Windows XP Professional?
- TC2.2.6 Does application successfully migrate from Windows 2000 Professional to Windows XP Professional?

2.3 Do not overwrite non-proprietary files with older versions

The application’s installation program must properly check to ensure that the latest file versions are installed. Installing an application must never regress any files that you do not produce or that are shared by applications that you do not produce.

Do not prompt the user to update a component unless a version update is actually required.

Application binaries must have valid file version information

Correct file version information has several benefits, including making it easier to meet the requirement of not overwriting files with older versions. Accordingly, *all executable images that you distribute must contain valid file version information*. When you display or get the properties of an executable (EXE, DLL, OCX, CPL, and so on), they must contain an accurate Product Name, Company Name, and File Version.

Implementation Details - 2.3

The **VerInstall** or **GetFileVersionInfo** APIs are recommended for performing the version check. **VerInstall** is easiest to use, because it uses **GetFileVersionInfo** internally and simplifies the process. Here is a simplified example of how to use **VerInstall**:

```
TCHAR szOldDir[MAX_PATH], szTempDir[MAX_PATH];
UINT cchTemp = MAX_PATH;
DWORD dwResult = VerInstallFile(0, TEXT("file.dll"),
TEXT("file.dll"), g_tszSetupDirectory, g_tszInstallDirectory,
szOldDir, szTempDir, &cchTemp);
if (dwResult != 0) {
    Error(dwResult);
}
```

Test Cases - 2.3

As defined in *“Designed for Microsoft Windows XP” Application Test Framework*:

- TC2.3.1 Does application not overwrite non-proprietary files with older versions?
- TC2.3.2 Do all application executable files have file version, product name and company name information?

2.4 Do not require a reboot inappropriately

In Windows XP, very few installation situations require a reboot. Reboots are unwelcome by customers and, in some situations, can make deploying applications difficult. The application must not require or suggest an unnecessary reboot during or after installation.

Some situations that require a reboot

- Installing a Windows Service Pack or authorized system redistributable may require a reboot.
- Installing a Graphical Identification and Authentication dynamic link library (GINA) requires a reboot. The GINA is a replaceable DLL component that is loaded by Winlogon. The GINA implements the authentication policy of the interactive logon model, and it is expected to perform all identification and authentication user interactions. For example, replacement GINA DLLs can implement smart card, retinal scan, or other authentication mechanisms in place of the standard Windows XP user name and password authentication. For more information on GINA, see 1.5 Support Fast User Switching.

Situations that do not require a reboot

- DLL registration.
- Updating a service component. If necessary, you must warn the user that certain services will be stopped while they are updated.
- Replacing an existing file that is in use by an application. You must give the user information about any open applications that have loaded the resource files you are updating so that the user can shut down those files and allow file replacement to occur without a reboot.

Also, for many components, you should install the components side-by-side or use **MoveFileEx** with the delay until reboot option to avoid this situation.

If you do require a reboot, you must prompt users and allow them the option of deferring the reboot.

You must also document the specific reason for the reboot, even if it is one of the specific items listed in this section.

Test Cases - 2.4

As defined in *“Designed for Microsoft Windows XP” Application Test Framework*:

- TC2.4.1 Does the installation finish without requiring a reboot?
- TC2.4.2 Can all Test Framework testing be completed without application requiring a reboot?

2.5 Install to Program Files by default

By default, the application must install into an appropriate sub-directory where the current user’s program files are stored.

- If you are using the Windows Installer, this folder is represented by the ProgramFilesFolder property in a Windows Installer-based package. (The ProgramFilesFolder property is a variable that exposes the path to the Program Files folder, and the Windows Installer sets that variable appropriately on all Windows platforms.)
- If you are not using the Windows Installer, the recommended method is to use the **SHGetFolderPath** API to retrieve the string represented by the CSIDL_PROGRAM_FILES value. On English-language systems, this folder is often C:\Program Files. However, do *not* hard-code that path, even for use on English systems, because it is not universal.

WHEN DOES THIS APPLY?

If you are upgrading a previously installed version of the application, it is acceptable to default to the directory on which that version exists.

If your application does not require installation (it executes without any files being installed onto the system), then this requirement is not applicable.

Considerations for shared components

In some cases, shared components must be placed in locations other than the application directory. This is described in the following section.

Test Cases - 2.5

As defined in “Designed for Microsoft Windows XP” Application Test Framework:

- TC2.5 Does application offer a default installation folder under “E:\Program Files?”

2.6 Install any shared files that are not side-by-side to the correct locations

Windows-based applications can share code, application, and component state in the registry, application-specific data in the file system, and Windows APIs that expose global namespaces. Sharing enables the efficient leverage of limited hardware resources and reduces the exposed front that Quality Assurance groups must test.

However, there are costs to sharing. Sharing causes applications to become interdependent upon one another, which introduces an element of fragility. In the extreme, applications that previously

worked might mysteriously start functioning oddly, or even fail. Typically, an application is dependent on a particular version or implementation of a shared component. If that shared component is upgraded (or downgraded) as a result of installing another application, the former application may break.

Windows XP, Windows 2000, Windows 98 Second Edition, and Windows Me enable side-by-side sharing, which minimizes this application fragility. Side-by-side sharing enables multiple versions of the same Win32 component to run at the same time in memory.

This means that applications can use the specific components that they were designed for and tested with, even if another application requires a different version of the same component. This enables developers to build more reliable applications, because they can choose the version of the component that they will use for their application, independent of other applications on the system.

The proper location for shared components that are not shared side by side depends on whether these components are shared across companies or by a single company:

- Shared components that are private to a single software vendor must be installed in one of two places: the common files directory, or the publisher's directory under the Program Files folder. Do not store these files in the System directory.
- New Control Panel items (CPLs) must be installed in the application directory on Windows XP.
- Services and device drivers must be placed in the System directory.
- OCXs and DLLs that are not side-by-side and are shared by multiple software vendors can be placed in the System directory to ensure backward compatibility with those applications.

When submitting your application, you must document any cases in which your software application writes to the System directory.

Implementation Details – 2.6

The proper location for shared components that are not shared side by side depends on whether these components are shared across companies or by a single company:

- Shared components that are private to a single software vendor must be installed in one of two places. Do not store these files in the System directory.

%CommonProgramFiles%\<company name>
-or-
%ProgramFiles%\<company name>\Shared Files

The common files directory can be accessed by passing `CSIDL_PROGRAM_FILES_COMMON` to the **SHGetFolderPath** API, or by using the Windows Installer CommonFilesFolder property. For more information about using Windows Installer Properties, see the *Windows Installer Programmer’s Reference* in the *Platform SDK*.

- Services and device drivers must be placed in the System directory.
- OCXs and DLLs that are not side-by-side and are shared by multiple software vendors can be placed in the System directory to ensure backward compatibility with those applications.
- New Control Panel items (CPLs) must be installed in the application directory on Windows XP. Register the path by adding a value under either of the following registry keys:

`HKEY_LOCAL_MACHINE\software\microsoft\windows
 \CurrentVersion\control panel\cpls`

—or—

`HKEY_CURRENT_USER\software\microsoft\windows
 \CurrentVersion\control panel\cpls`

EXAMPLE A name/value pair under this key:

`MyCpl = "%ProgramFiles%\MyCorp\MyApp\MyCpl.cpl"`

Test Cases - 2.6

As defined in “*Designed for Microsoft Windows XP*” Application Test Framework:

TC2.6 Does application install shared files only to correct locations?

2.7 Support Add or Remove Programs properly

The application must supply your product’s name, the location of your application’s uninstaller, and so on, so that the Add or Remove Programs item in Control Panel can obtain information about the application as needed. You can write this information directly to the registry during install or, if you are using an installation system based on the Windows Installer service, you can set these values by using properties in the Windows Installer-based package.

WHEN DOES THIS APPLY?

This requirement is not applicable for an application that does not install — that is, if it executes without installing any components, writing to the registry, modifying the system, or leaving any files on the system other than user created files.

Uninstall:

The application’s uninstaller must correctly and fully remove the application.

Except as noted later in this section, the application must remove the following:

- All non-shared application files and folders.
- Shared application files whose reference-count (refcount) reaches zero.
- Registry entries, except for keys that might be shared by other programs.
- All shortcuts from the Start menu that the application created at the time of installation.
- The uninstaller itself (unless it is a shared component).

When submitting your application, you must document any files not created by users that the application's uninstaller leaves behind.

The removal must be clean enough to allow the application to be reinstalled later. User preferences may be considered user data and left behind, but an option to do a completely clean removal should be included.

In general, all user data should be left on the system after removal. If your application removal is about to remove user data, the user must be prompted for confirmation.

An unprivileged user may attempt to remove the application. If a Limited User cannot uninstall the application, the uninstall must degrade gracefully.

Implementation Details – 2.7

The application must supply all the information in the following table so that the Add or Remove Programs item in the Control Panel can obtain information about the application as needed. You can write this information directly to the registry during install or, if you are using the Windows Installer service, you can set these values by using properties in the Windows Installer-based package.

The registry values must be written under the following key:

```
HKEY_LOCAL_MACHINE
  \Software
    \Microsoft
      \Windows
        \CurrentVersion
          \Uninstall
            \{ProductCode}
```

Registry value	Type	Related Windows Installer Property	Contains
DisplayName	REG_SZ	ProductName	Display name of application
UninstallString	REG_EXPAND_SZ	N/A	Full path to the application’s uninstall program, including the complete command line used to carry out your uninstall program
InstallLocation	REG_EXPAND_SZ	ARPINSTALLLOCATION	Full path where application is located (folder or .exe)
Publisher	REG_SZ	Manufacturer	Publisher/Developer of application
VersionMajor	DWORD	ProductVersion	Major version number of application
VersionMinor	DWORD	ProductVersion	Minor version of application

NOTE Property names are case sensitive.

You can also provide additional properties to present in the Add or Remove Programs item, such as product ID, online support information, and so on. See the Microsoft Platform SDK for full details.

Test Cases – 2.7

As defined in “*Designed for Microsoft Windows XP*” Application Test Framework:

- TC2.7.1 Does installation add all necessary entries to the registry?
- TC2.7.2 Does uninstalling application as Owner remove and leave all the correct files and registry settings?
- TC2.7.3 Does uninstalling application as User1 either degrade gracefully or both remove and leave all the correct files and registry settings?
- TC2.7.4 Can application be reinstalled after uninstalling it?

2.8 Support “All Users” Installs

Applications are often used by more than one user on the computer. Your installer must default to “all users” or provide an “all users” installation as an option. For example, an installer might default to the option of installing the application only for the current user but the application must provide an option to install for all users.

An unprivileged user may attempt to install the application. If a limited user cannot install the application or cannot install for “all users,” the installation must degrade gracefully.

Test Cases – 2.8

As defined in “*Designed for Microsoft Windows XP*” Application Test Framework:

- TC2.8.1 Does application default to an “all users” installation or provide an “all users” installation as an option when installed by Owner?
- TC2.8.2 Does application default to an “all users” installation or provide an “all users” installation as an option when installed by User1?

2.9 Support Autorun for CDs and DVDs

For applications distributed on CD, DVD, or other removable media that support Autorun, the first time the disc is inserted, the application must use Autorun to run the application or prompt the user to install. In the case of applications distributed on multiple discs, subsequent discs must either use the Autorun feature or continue installation without prompting the user to press a key or take other action when the CD has been inserted.

It is not acceptable to require the user to use Start/Run to launch the installation from the CD or DVD.

After the application has been successfully installed, re-inserting the CD or DVD in the drive must not cause installation to automatically begin again. It is acceptable to ask users if they want to update or change their installation choices.

Test Cases – 2.9

As defined in “*Designed for Microsoft Windows XP*” Application Test Framework:

- TC2.9.1 Does application’s installer start by way of Autorun?
- TC2.9.2 Does application’s installer correctly detect that application is already installed and avoid restarting the installation?

3.0 Data and Settings Management

Summary of Data Requirements

Customer Benefits

Microsoft® Windows XP provides an infrastructure that supports state separation of user data, user settings, and computer settings.

Applications that use this infrastructure correctly offer the following benefits:

- Applications do not fail when run by Limited Users (non-Administrator), allowing family or friends to share a computer safely and easily.
- Parents can allow children to use the computer without giving them administrative privileges, which would give the child unrestricted access to modify the computer.
- Users can back up their individual documents and settings easily without needing to back up application and operating system files.
- Multiple users can share a single computer, each with his or her own preferences and settings.
- Applications are less likely to prevent Fast User Switching from operating correctly and efficiently.

Data and Settings Requirements List

- 3.1 Default to the correct location for storing user-created data
- 3.2 Classify and store application data correctly
- 3.3 Deal gracefully with access-denied scenarios
- 3.4 Support running as a Limited User

Data and Settings Management Requirements

3.1 Default to the correct location for storing user-created data

User-created files must be stored in the user’s My Documents (or descendant) folder. For imaging files, it is recommended to use My Pictures in place of My Documents. Similarly, use My Music for audio files.

The first time a user runs an application, the application’s File Open and Save dialogs must default to the user’s My Documents (or descendant) folder the first time these dialogs are called. On subsequent calls to these dialogs, it is recommended to default to the previously selected path.

Application data, such as user preferences, application state, temporary files, and so on, must not be stored within My Documents. The correct locations for these items are defined in requirement 3.2.

Calling the Common File Open/Save with no parameters will default to the My Documents folder. To target the My Documents folder directly, you must pass CSIDL_PERSONAL to **SHGetFolderPath**. In addition, if the user had selected a new location for saving files from the Common File Open/Save dialog, Common File Open/Save will be smart and default to the user selected location the next time.

The benefits of using the My Documents folder as the default location for data storage are:

- All users (including those with restricted account types) have write access to this location.
- Users have one familiar place to organize and store all their data.
- Data sharing is facilitated between applications because all applications using Common File Open can easily access the My Documents folder.
- My Documents is an abstracted location and can be redirected to the network transparently by an administrator.
- My Documents is available on the Start menu.

Retrieving the path to My Documents: The only acceptable way to do this is by passing CSIDL_PERSONAL to the **SHGetFolderPath** API. You must not use hard-coded paths or registry entries.

```
TCHAR szMyDocs[MAX_PATH];  
...  
hr = SHGetFolderPath(NULL, CSIDL_PERSONAL, NULL, 0, szMyDocs)
```

Implementation Details - 3.1

SHGetFolderPath is exported from Shfolder.dll and behaves consistently across Windows 95, Windows 98, Windows Me, Windows NT 4.0, Windows 2000, and Windows XP. Shfolder.dll is included with Windows XP, Windows NT 4.0 Service Pack 4 (and later), Microsoft Internet Explorer 5, Windows 2000, Windows 98 Second Edition, and Windows Me. Shfolder.dll is a redistributable component that contains support for CSIDL_PERSONAL as well as many other special folders. (For more information, see "S1.0 Game Experience.")

Shfolder.dll is installed by the Windows Installer redistributable. Software vendors are encouraged to redistribute this component as much as possible to enable this support on versions of Windows operating systems earlier than Windows XP. Windows XP includes this DLL file as a protected system file and, as such, it cannot be replaced on Windows XP or later.

To help ensure that the application can run on Windows 9x and Windows NT 4.0, as well as Windows 2000 and Windows XP, always link to the **SHGetFolderPath** implementation in Shfolder.dll. Windows 2000 and Windows XP natively implement

SHGetFolderPath in Shell32.dll, but older versions of Windows may not include **SHGetFolderPath** in Shell32.dll.

NOTE The user may need to move the My Documents folder (and other system documents folders), or they may be moved in a system update. Do not depend on the My Documents or other such folders being in the same location next time the application is launched. This issue can arise if you store the full path to files in the My Documents folder in an MRU or other application specific place.

Test Cases – 3.1

As defined in “*Designed for Microsoft Windows XP*” Application Test Framework:

- TC3.1.1 Does application offer a correct location for opening User1’s user-created data?
- TC3.1.2 Does application offer a correct location for saving User1’s user-created data?
- TC3.1.3 Does application offer a correct location for opening User2’s user-created data?
- TC3.1.4 Does application offer a correct location for saving User2’s user-created data?

3.2 Classify and store application data correctly

Applications typically work with two fundamental types of documents: those that are created by the user, and those that are created for use by the application itself. For user-created data, please refer to the guidelines in section 3.1. Section 3.2 applies directly to the data that is created by an application to store user-specific information. This data is used by the application only, and is never intended to be accessed directly by the user.

By storing this application-specific data in one of the several valid locations as described by this specification, you make it possible for multiple people to use the same computer without corrupting or improperly modifying each other’s data. The specification provides several valid locations and you are free to choose the location that works best for your needs.

A clear benefit to the developer is that compliance with this requirement can actually result in fewer lines of code. There is a single API call, **SHGetFolderPath**, that enables you to determine the correct location in which to store the user’s data and the user-specific application data.

Classifying and storing application data according to the guidelines in this requirement provides these benefits:

- It enables multiple family members to share a computer and helps enable Fast User Switching.
- It enables business-related operations such as roaming, off-line storage, and allowing the operating system and its applications to be secured.

- It ensures a consistent and abstracted location for user data, enforces per-user separation of application data.
- It is one of the key factors in enabling remote use of the application.

Application data includes user preferences, application state, temp files, and so on.

IMPORTANT

For files that the user can open (such as documents, music, pictures, and so on), use the guidelines in section 3.1.

This section identifies the valid file folders and the valid registry locations that applications must use for this data, and gives guidance on how to choose which of these locations are best used in different circumstances. The choice of valid locations to use is left to the software developer.

Classify application data into the following categories:

- Per user, roaming
- Per user, non-roaming
- Per computer (non-user specific and non-roaming)

NOTE There may be more than one category for the different application data stored by your application.

For applications not intended to be used in a domain environment (most games and home products for example), classifying the application data as per user, non-roaming might be an appropriate choice.

It is best to use application data file folders rather than the registry for storing application data in excess of 64K. The registry is an acceptable choice for small amounts of data. At installation time, try to store less than a total of 128K across HKEY_CURRENT_USER (HKCU) and HKEY_LOCAL_MACHINE (HKLM).

To comply with this specification, store application data files appropriately as either common or per-user. That is:

- In a subfolder of either the common application folder (identified by CSDL_COMMON_APPDATA), or
- In the user profile folders: application data (CSDL_APPDATA) or local application data (CSDL_LOCAL_APPDATA).

The subfolder to create to store user data files in is:
[company name]\[product name]\[version].

Using the Registry

Applications may also use the registry to store read/write application data and configuration files.

- The HKCU registry hive is appropriate for storing small amounts of data (approximately 64K) and for policy settings that are per user.
- Avoid writing to HKLM during runtime, because limited users have read-only access to the entire HKLM tree by default. In addition, HKLM does not support roaming.
- Larger, file-based data should be placed in the Application Data folder. For example, Internet Explorer’s Temporary Internet Cache is stored within the user profile and not in the registry.
- At installation time, the application must not store more than a total of 128K across HKCU and HKLM.
Note that HKEY_CLASSES_ROOT is excluded.

Implementation Details - 3.2

Using Application Data Folders. Once you have decided how to classify your data, you can use **SHGetFolderPath** to retrieve the corresponding folder locations.

The CSIDL values described here provide a consistent, unified way to access the physical paths to the desired folder locations, independent of the operating system. The preferred API is **SHGetFolderPath**, because it behaves consistently across all versions of Windows. To access the path for application data, applications should call **SHGetFolderPath** with the appropriate CSIDL and then append *[company name]\[product name]\[version]* to the returned path. Specifically:

To retrieve the CSIDL_APPDATA path:

```
TCHAR szAppData[MAX_PATH];  
...  
hr = SHGetFolderPath(NULL, CSIDL_APPDATA, NULL, 0, szAppData);
```

When storing application data in the user profile, applications must use the following hierarchy under the Application Data file structure:

```
[User Profile]\  
  Application Data\  
    [company name]\  
      [product name]\  
        [version]\  
          [file or folder]
```

Data Type	Folder CSIDL	Folder Location
Per user, roaming	CSIDL_APPDATA	[user profile]\Application data
Per user, non-	CSIDL_LOCAL_APPDATA	[user profile]\Local

roaming		Settings\Application data
Per computer (non-user specific and non-roaming)	CSIDL_COMMON_APPDATA TA	All Users\Application data

To comply with this specification, applications must classify and store data appropriately as either common or per-user. That is, either CSIDL_COMMON_APPDATA or one of the user profiles: CSIDL_APPDATA or CSIDL_LOCAL_APPDATA.

CSIDL_APPDATA

This folder will be enabled for roaming with the user profile. Use this folder to store all user-specific application preferences. For example, if a user can specify a custom dictionary to be used in the application, you would store it here. That way, if a user roams from computer to computer, the dictionary will roam with him or her. This also allows other users to have their own custom dictionaries.

CSIDL_LOCAL_APPDATA

This folder is for application data that does not roam. As it is still part of the User profile, this is still per-user information. Application data that is computer-dependent, such as user-specified monitor resolution, must be stored here.

This data must not roam because different computers are likely to have different monitors. In addition, large blocks of data that can easily be recreated and temporary files must be placed here to minimize download time that is incurred when roaming.

EXAMPLE Internet Explorer keeps its cache of downloaded .html/.gif pages here so that they don't roam with the user. However, the smaller cookie and history lists are stored in CSIDL_APPDATA so that they do roam.

CSIDL_COMMON_APPDATA.

This folder should be used for application data that is not user specific. Note that a limited user will only have read privilege for files in this folder, except for the files that user created. If users need to have write access to the common files, then during installation the application must create a sub-folder of CSIDL_COMMON_APPDATA with “Modify” privilege for all users.

EXAMPLE An application may store a spell-check dictionary, a database of clip art or a log file in the CSIDL_COMMON_APPDATA folder. This information will not roam and is available to anyone using the computer.

Additional Considerations

- Files may be shared in the User Profile\Application Data folder. Multiple computers may use them simultaneously with different instances of the application. The data may also be used by multiple applications, for example, applications in a productivity suite.

Applications should get a write exclusive on the file only when absolutely necessary. For example, applications using **CreateFile** should only specify GENERIC_WRITE when a write is required, but they should always set FILE_SHARE_READ.

- Paths returned by **SHGetFolderPath** are valid Win32 file system names that may contain spaces and may be in the universal naming convention (UNC) format.
- **PathAppend()** and **PathCombine()** APIs can be used to concatenate the relative path information onto the paths returned by **SHGetFolderPath**. For example:

```
PathAppend(szAppData, "Company\Product\File.txt")
```

- Any user can write into the All Users\Documents location. However, by default, only the creator of the document (and administrators) will be able to subsequently modify the document. All other (non-administrator) users will have read-only access to the document by default.

If an application requires all regular users to have write access to a given application-specific subdirectory of CSDL_COMMON_DOCUMENTS, the application must explicitly modify the security on that subdirectory during application setup. The modified security must be documented.

Test Cases – 3.2

As defined in “Designed for Microsoft Windows XP” Application Test Framework:

TC3.2.1	Does application store less than 128K of application data in the registry for User1?
TC3.2.2	Does application store configuration data for User1 only in acceptable folders?

3.3 Deal gracefully with access-denied scenarios

By default on Windows XP, restricted user accounts (for example, Limited Users on Home Edition) cannot write to per-computer locations such as HKLM and the Windows directory. Only applications that classify and store data correctly, as described earlier, will be able to avoid access-denied errors and run successfully in this secure environment.

There are, however, legitimate scenarios in which access-denied errors are encountered by applications that classify and store data correctly.

EXAMPLE Appropriate cases for access-denied scenarios:

- An unprivileged user may attempt to run an administrative application designed to modify system-wide settings. In this case, the user is allowed to run the application, but cannot carry out any system-wide modifications.
- An unprivileged user may run an application that allows users to modify objects based on permissions. In this case, a user is allowed to modify objects that he or she owns, but not objects owned by the administrator or other users.
- An unprivileged user may direct an application to save per-user data in a per-computer location, for example, by choosing the System directory as the destination for a File\Save operation.
- An unprivileged user may attempt to install the application. If a limited user cannot install the application, the installation must degrade gracefully.

In these cases, the application must degrade gracefully when the access-denied error is encountered. Graceful handling can be accomplished by:

- **Disabling the operation.** This is the approach taken by System Settings in Control Panel. Users cannot set system-wide environment variables, but they can set their own environment variables. Thus, when a user launches this applet, the system-wide environment variable option is disabled. When an administrator launches the applet, the system-wide environment variables can be modified.
- **Displaying an appropriate error message.** For example:
“You must be an administrator to perform this operation.”
“You only have permissions to view the properties of this object.”
“You must have X privilege to perform this operation.”
“You must have write access to (file name) to perform this operation.”

Test Cases – 3.3

As defined in “*Designed for Microsoft Windows XP*” Application Test Framework:

TC3.3.1	Does application prevent User1 from saving to the Windows system folder, E:\Windows?
TC3.3.2	Does application prevent User1 from modifying documents owned by User2?
TC3.3.3	Does application prevent User1 from modifying system-wide settings?
TC3.3.4	Does application’s installer either allow User1 to install application or degrade gracefully if the installation fails?

3.4 Support running as a Limited User

Applications must not require users to have unrestricted access (for example, Administrator privileges) to make changes to system or other files and settings. In other words, the application must function properly in a secure Windows environment. Complying with the

previous requirements in this section will help to ensure that the application meets this requirement.

An application that does not install (executes without installing any components) must still support use by a Limited User.

A secure Windows environment is defined as the environment exposed to a Limited (non-Administrator) user by default on a clean-installed NTFS system. In this environment, users can only write to these specific locations on a local computer:

[Note 1]

- Their own portions of the registry (HKEY_CURRENT_USER)
[Note 2]
- Their own user profile directories (CSIDL_PROFILE)
- A Shared Documents location
(CSIDL_COMMON_DOCUMENTS) [Note 3]
- A folder that the user creates from the system drive root

However, applications defaulting to use of these folders do not comply with the other requirements of this section.

Users can also write to subkeys and subdirectories of these locations. For example, users can write to CSIDL_PERSONAL (My Documents) because it is a subdirectory of CSIDL_PROFILE. Users have read-only access to the rest of the system.

NOTES

[1] Applications can modify the default security for an application-specific subdirectory of CSIDL_COMMON_APPDATA. This may provide an additional location to which users can write for a given application.

Any modification of the default security for an application-specific subdirectory of CSIDL_COMMON_APPDATA must be documented when submitting your application.

[2] Users cannot write to the following subsections of HKCU:

 \Software\ Policies
 \Software\Microsoft\Windows\CurrentVersion\Policies

[3] By default, users cannot write to other users' shared documents; they can only read other users' shared documents. Applications can modify this default security on an application-specific subdirectory of CSIDL_COMMON_DOCUMENTS.

Any modification of the default security on an application-specific subdirectory of CSIDL_COMMON_DOCUMENTS must be documented when submitting your application.

This requirement does not apply to all features.

WHEN DOES THIS APPLY?

When the major features of the application can be successfully run by a non-privileged user, minor features are allowed to fail gracefully. These minor features must not be installed by any default mechanism (for example, a minimal or typical install) other than a complete install and must not be considered important for the operation of the program. Examples of such minor features include components necessary to support legacy file formats.

Limited Users cannot perform several system administration functions such as disk defragmentation, backup/restore, changing system time, and so on. When most of the primary functionality of an application is system administration, the application must still run from a Limited User account and inform the user why none of the features can be used.

For any feature that a limited user cannot use, when submitting your application you must document what objects need to be opened for that feature to work, such as file system, registry keys, and so on.

When a limited user can't use a feature, the application must degrade gracefully.

Test Cases – 3.4

As defined in *“Designed for Microsoft Windows XP” Application Test Framework:*

TC3.4 Does application support running as User1, a Limited User?

Section 2: 'Designed For Windows XP - Optimized' Requirements

The 'Designed For Windows XP - Optimized' Requirements section includes incremental specifications to the Designed for Windows XP requirements. Complying with these is not required to achieve the "Designed for Windows XP" logo. However, compliance with one of the following three groups of requirements (which apply to your application) will result in a product achieving the Designed for Windows XP - Optimized Status.

S1.0 Optimized for Game Experience

S2.0 Optimized for Windows XP

S3.0 Optimized for Accessibility

S4.0 Optimized for .NET

S3.0 Optimized for Enterprise

S1.0 Optimized for Games

Summary of the Optimized for Game Experience Guide

"*Designed for Microsoft Windows XP*" Application Specification defines the technical requirements for applications to earn the "Designed for Microsoft Windows XP" logo. The guidelines outlined in this section are in addition to the "*Designed for Microsoft Windows XP*" Application Specification as defined in the previous sections.

Due to the inherent complexity of games and their reliance on hardware functionality, to receive the 'Designed For Windows XP - Optimized' status, it is necessary to adopt these additional requirements so as to ensure a more consistent and stable user experience.

Additional information about these guidelines may be available at <http://www.microsoft.com/winlogo/software/swfuture.asp>.

Game Experience List

S1.1	General game playing experience
S1.2	Do not install or depend on 16-bit files
S1.3	Do not write to Config.sys, Autoexec.bat, Win.ini, or System.ini files
S1.4	Provide both manual and automatic install options
S1.5	Allow installation termination
S1.6	Running from CD or DVD
S1.7	Games must be rated
S1.8	3-D and graphics support
S1.9	Audio support
S1.10	Device support
S1.11	Network support

Game Experience Guide

S1.1 General game playing experience

Game applications should meet the requirement to "Install Shortcuts Correctly" in the "Future Requirements" subsection.

The application should make sensible default choices for three-dimensional (3-D) device, resolution, and color depth without asking the user.

A game must select the default choice for the user. An example of a sensible default choice would be that, if 800x600 resolution gives the best user experience and is supported by the game and the user's computer system, it should be selected automatically for the user. It is acceptable to provide an option for a user to override these options. This requirement can be met either in the installation or on the first run of the game.

The user’s computer performance characteristics should be analyzed. The options for the game should be set up in such a way that the user can play the game immediately without adjusting game playing options, display, shadowing, animations, 3-D effects, and so on, to get acceptable performance from the game on his or her computer.

Verifying that the computer is above the minimum requirements is the developer’s responsibility and is not part of this requirement. This requirement can be met either in the installation or on the first run of the game. If you can install the game on a computer that is at the minimum requirement and know that this computer will give the user a playable frame-rate, your application passes this requirement.

During either install or first run, the user’s input devices should be polled. If more than one exists, then the best device for the game must be selected. The device selected should be reasonably configured with buttons that are set up to play the game. An example of the best available input device would be that, if a user has a steering wheel and a joystick and installs a driving game, the steering wheel would be chosen as the best available input device.

The game should use **LoadLibrary** or **CoCreateInstance** to give itself the option of failing more gracefully instead of displaying the system “missing DLL” dialog box.

If the game uses the **LoadLibrary** function to load any system components (that is, Microsoft Direct3D®, Microsoft DirectInput®, and so forth), do not hard-code paths.

The game should perform its basic operational functions over at least four hours without any problems. The game must still function after crossing the date change barrier.

If any system components that the game relies upon change then another first-run analysis of the computer should be performed. For example, when a new input device or display adapter is installed another analysis should be executed.

S1.2 Do not install or depend on 16-bit files

Because 16-bit code does not run on 64-bit hardware, any dependency on 16-bit code must be avoided. This will allow your customers more hardware choices. However, running on 64-bit hardware is not a requirement.

The application should not crash or fail to execute on 32-bit edition Windows XP because of a dependency on 16-bit code. The application must not read, install or depend on 16-bit files.

S1.3 Do not write to Config.sys, Autoexec.bat, Win.ini, or System.ini files

These files are not used on any Windows operating system based on NT technology, including Windows XP. Some users remove these files

from their computers. The application must not read from or write to Win.ini, System.ini, Autoexec.bat, or Config.sys on any Windows operating system. The application should not have 16-bit dependencies that make reading and writing to these files necessary.

S1.4 Provide both manual and automatic install options

Near the start of installation (after any option license agreement), provide the user with the option of selecting either Automatic or Manual installation. The Automatic installation will make as many logical choices as possible for the user.

The Manual installation may take the user to another dialog box where the user can choose any number of installation types: custom, typical, minimal, and so on. These installation types may require feedback from the user about where to copy the files, which monitor to use, which game playing devices to use, which folder to install the shortcuts, and so on.

The Automatic installation will progress through the installation, acting as if the user had selected each default installation option from the manual installation, so as to minimize the number of interactions. Each manual installation dialog box for which the automatic install selects a default should not be displayed. Examples of dialogs that the user would not see are installation folder, shortcut location, installation of desktop icon, installation of DirectX, and so on.

Each dialog box of the installation should have a default button highlighted.

Both the Automatic and Manual installations should display a warning message if there is not enough space on the hard drive for the installation.

The Automatic installation should be the default installation.

S1.5 Allow installation termination

The user should be able to cancel out of an installation at any time before the installation completes. For each dialog box of this install process, make sure that your application can stop the installation.

The installation should confirm that the user wants to abort the install and did not accidentally press the wrong key. Before stopping an installation, ask if this is really what the user wants to do. If the user says "No," then the installation should continue.

When a user stops the installation, the install program should clean up the failed installation and tell the user how to restart the installation.

S1.6 Running from CD or DVD

If the game requires a CD or DVD for game play even with the game fully installed on the hard drive, launching the game from the Start menu\Games folder should display an "Insert CD or DVD" message if the CD or DVD is not in the drive. When the CD or DVD is inserted, then the game should continue to run without requiring the user to restart the game.

The game must be able to be played on a different CD or DVD drive than the one from which it was installed. The program must be able to load from any valid drive. When the program is first run, it should search all CD or DVD drives available for the game CD or DVD if needed. It is not a requirement that the game supports a CD Jukebox.

The game must work even in situations where Red Book audio is not available for a particular drive (for example, external drives). If the game needs to get audio files from the CD or DVD, the audio should still work even when run from an external CD drive.

The drive letter used for CD drives might not remain the same as when the application first installed. This can even happen inadvertently, because it is now easy for a user to add hardware or make other configuration changes without realizing all of the consequences.

Once an application is installed, it should have no problem when the CD drive letter has changed. If it requires the presence of the CD for updates, validation, additional data, and so on, it must be able to use the new drive letter. The user must not be required to modify the registry, reconfigure hardware or system configuration, or reinstall the application to use primary functionality.

S1.7 Games must be rated

The game must be rated by and adhere to guidelines specified by the Entertainment Software Ratings Board (ESRB) if the game is to be distributed in North America and by the appropriate ratings for the country in which the game is distributed.

S1.8 3-D and graphics support

If the game uses Direct3D, it should be the default method for displaying graphics.

If the game employs 3-D graphics and runs in full-screen mode, all screen resolutions enabled in the game should be supported by the computer. If it runs in windowed mode, it should properly handle a range of resolutions. A friendly error dialog box must appear for any resolution that is not supported.

The game must not rely on the presence of Microsoft DirectX® color-keyed textures. Alpha channels and true-color textures are strongly

preferred, and products must not crash simply because the hardware does not support the obsolete methods.

The game must not rely on the presence of DirectX palette textures. Alpha channels and true-color textures are strongly preferred, and products must not crash simply because the hardware does not support the obsolete methods.

The game must not use retained mode or the ramp-mode rasterizer.

If the product employs 3-D graphics, it should support hardware acceleration. You should not attempt 3-D graphics without supporting hardware acceleration.

If the game supports 3-D graphics hardware acceleration then it must support cards from at least 2 chipset manufacturers.

If the game uses Microsoft DirectDraw®, it must not use device-independent bitmap (DIB) sections or other legacy methods.

If the game changes the resolution or colors to play the game in full-screen mode, it must restore the original resolution and color depth when the game exits.

If it changes the resolution or color-depth, when the game exits, it must restore the original resolution and color depth.

The game must not rely on Hardware Cursor Support.

S1.9 Audio support

If the game uses Microsoft DirectSound®, it should be the default method for providing sound.

The game should use the DirectSound API to handle all calls and not call the cards directly to use specific features of specific cards.

On systems with more than two audio output channels, audio must emanate from appropriate speakers. For games that use 3-D audio, this means that sounds should pan around the listener using all available speakers. For games that do not use 3-D audio, audio may, but need not, emanate from the rear speakers.

The game must not disturb volume settings. If the game can adjust its audio volume, settings must be unchanged on exiting the Windows Volume Control.

If audio is not essential to the operation of the game, it must run on a computer without a sound card installed.

S1.10 Device support

If the game uses DirectInput, it should be the default method for input.

The game must support game playing devices on more than ID1. Game playing devices on ID2 or higher must be supported.

A method should be provided for the user to easily select which device he or she wants to use to play a game (that is, switching from one joystick to another or from a joystick to the keyboard and mouse).

On platforms that support multiple keyboards, the game must work properly when there are two keyboards plugged into the system. Because the limitation on the number of keys pressed simultaneously is normally a hardware limitation, it can be overcome by using multiple keyboards. The game must not crash, stop responding, or lose data when multiple keyboards are used to cause more simultaneously pressed keys to be reported than would be possible with one keyboard.

The game must not fail when used with mice with more than three-mouse buttons.

S1.11 Network support

If the game has an online component, it should recognize when it is in its offline state. For example, the game must not spend minutes trying to reach an unreachable master server.

All non-working networking options should be detected and disabled from the list of choices presented to the user.

During lengthy network transactions, the user should be able to cancel operations at any time.

S2.0 Optimized for Windows XP

Summary of the Optimized for Windows XP Guidelines

Overview

The Optimized for Windows XP guidelines define what an application should do to exploit the new features of Microsoft® Windows XP and provide the best customer experience including:

- Easy to read high quality icons
- AutoPlay of their content
- An organized Control Panel
- Sharing a single publicly routable IP address between several PCs

Additional information about these guidelines may be available at <http://www.microsoft.com/winlogo/software/swfuture.asp>.

Optimized for Windows XP Requirement List

S2.1	Integrate the New Visual Styles
S2.2	Author high-quality icons
S2.3	Optimize for Fast User Switching
S2.4	Write AutoPlay handlers
S2.5	Follow Control Panel Taxonomy
S2.6	Enable traversing Network Address Translation

Optimized for Windows XP Guide

S2.1 Integrate the New Visual Styles

Windows XP introduces a new look for Windows. This new look, the first major overhaul since Windows 95, is a dramatically different visual style for the operation system. Windows XP also introduces a new architecture for how the user interface for Windows is drawn. This architecture allows users to select a visual style for Windows. Users may choose the new Windows XP Style, or the Windows Classic Style. The architecture provides the ability for Microsoft to create more visual styles and update the operating system with more choices for the user. For compatibility reasons, applications will only get part of the new visual styles and the new architecture by default. In order for an application to look like it was designed for Windows XP, and so that it can continue to look correct no matter what visual style the user has selected, application developers must integrate the new visual style in their application.

Applications should use version 6 of the Windows Common Controls (comctl32.dll). This will ensure that all Windows controls are displayed with the current visual style chosen by the user.

Applications must supply a manifest for their application that specifies using version 6 of the Windows Common Controls or greater. More

information about how to supply a manifest for your application is available in the Microsoft Platform SDK, "Enabling Updated Common Controls, Visual Styles, and Themes."

If the application hosts third-party components at runtime that are not tested by the application developer to ensure that they operate correctly with the new visual styles, then the application cannot simply supply a manifest and have it apply to the application and all the components the application uses. In this case, applications should follow the steps detailed in the Microsoft Platform SDK under the "Enabling Updated Common Controls in an Application with Extensions, Plug-Ins, or Control Panels" section.

Applications that do not completely replace the caption bar appearance must use a standard Windows title bar, including the standard buttons. Applications should not render anything custom into the standard caption area.

Applications that render their visuals so that they appear with the same visual style as the operating system must use the Theme Manager to render the visuals on the applications behalf. Applications should provide a default user experience consistent with the Windows look and feel.

More information about integrating the Windows XP visual styles in your application can be found in the Microsoft Platform SDK, "Using Windows XP Visual Styles."

S2.2 Author high-quality icons

Another aspect of the new look of Windows introduced by Windows XP is the use of a more friendly yet sophisticated icon style.

These icons take advantage of the capabilities found on most new systems today, specifically 32-bit per pixel display devices. Windows XP now adds support for 24-bit color icons with 8-bit masks. These capabilities allow designers to create icons that will display with smooth curves and subtle shadings and anti-alias on any background color- no more jaggy icon edges.

Applications will need to update their icons to take advantage of these capabilities in order to appear that they were designed for Windows XP. Among other locations, these new icons are displayed in the shell when Icon, Thumbnail, or the new Tiles view is selected.

Designers should author 16x16, 32x32, and 48x48 icons for their applications and devices in this new color depth by using the new icon tools and guidelines. Applications that use icons on toolbars should author 24x24 and 16x16 icons for toolbar use.

For guidelines on how to create high-quality icons, follow the five steps described in the *Windows XP Icon Design Guidelines*.

Implementation Details – S2.2

"Windows XP Icon Design Guidelines" whitepaper, available at
<http://www.microsoft.com/hwdev/windowsxp/downloads>

S2.3 Optimize for Fast User Switching

Computers in home environments are often shared among the members of the household and in many businesses computers are shared among workers. Windows XP introduces a new way to share a computer: Fast User Switching. In Windows XP, it is not necessary for a user to log off the computer when another user wants to access their account. Instead, the user's account is always logged on, and users can switch quickly between all open accounts. Many accounts can be open simultaneously on one computer. Applications designed for Windows XP should be enabled to run in a Fast User Switching environment.

Applications should be able to run simultaneously across multiple Windows sessions. This means every user logged on to the system should be able to run the application at the same time as other users are running the application.

The Data and Setting Management requirements, 3.1-3.4, are fundamental to enabling applications to run simultaneously across multi-user sessions. By following these requirements, applications are ensured that each user's data is kept separate from other the data of other users.

In many cases, applications will not have to do any additional development to protect data when multiple sessions are running. If your application must share data or other resources across different users or session, then your application must take additional precautions. In this case your application must have a multi-user locking scheme in place to prevent data corruption, data loss, or resource conflicts.

It is important that applications use as few shared resources as possible and that they free the resources as quickly as possible. For example, applications that are waiting for a device to connect on a serial port, should release the port when another session is active and attempt to re-acquire the port when their session becomes active. Applications that prevent instances from running across multiple sessions as their multi-user locking scheme are not considered Optimized for Windows XP Applications.

Applications must conserve system resources when a different session is active. Applications should not play any sounds or write to the display when another user has the active session. In addition applications should not perform any unnecessary operations.

Consider the following situation: An application checks a status every 30 seconds, and then based on the status updates some information on the display. When the application is running in a non-active

session, it is clear that no display update should take place. However since no display is going to be updated, the status check itself should not occur. Waking the application every 30 seconds to perform a task whose result can not be shown, is wasteful of system resources.

If an application includes a service, the service must ensure that interactions with the user occur with the correct user. Services should not assume that session 0 is the current desktop session. On Windows XP the active session can have any session number. Use the function **WTSGetActiveConsoleSessionID()** to identify the active session

Implementation Details – S2.3

Applications can track if they are the active session using the WM_WTSSESSION_CHANGE message. Applications must register to receive this message by calling **WTSRegisterSessionNotification** function. See the Microsoft Platform SDK for more information.

S2.4 Write AutoPlay handlers

AutoPlay is a Windows feature that detects:

- Content such as pictures, music, or video files on removable media and removable devices.
- Blank CDs, audio CDs, commercial DVD's, etc.
- Devices such as MP3 players, digital camera's, etc.

AutoPlay can automatically launch applications to play or display content or handle events and devices. It can also allow the user to select among registered handlers if there is more than one for the event or device.

When an application enables an important scenario for a device or type of media, the application should either register itself to handle specific events for one or more AutoPlay **DeviceHandlers**, or for devices exposing a volume interface the application should register under the appropriate existing **EventHandler** (the AutoPlay documentation contains information on the existing **EventHandlers**).

WHEN DOES THIS APPLY?

This requirement does not apply if the application won't have tasks for specific devices, types of devices, or types of content.

An application can provide registration information at many levels, depending on its requirements. The following is a list of the potential registration settings an application may need to make:

DeviceHandlers
EventHandlers
Handlers

Consult the AutoPlay documentation for more details.

Implementation Details – S2.5

For information about how to implement AutoPlay handlers, see “Preparing Hardware and Software for Use with AutoPlay” in the MSDN: <http://msdn.microsoft.com/library/default.asp>

S2.5 Follow Control Panel Taxonomy

Control Panel now supports categorization of CPL items. For information, see *Windows XP Visual Specification Guide*, available at <http://www.microsoft.com/hwdev/windowsxp/downloads/>.

A Control Panel item should request registration in a particular category by adding a DWORD entry for the item in the Extended Properties of Control Panel in the registry.

Implementation Details – S2.6

A Control Panel item can now request registration in a particular category by adding a DWORD entry for the item in the registry. The following code snippet shows how the Accessibility item would register itself:

```
HKEY_LOCAL_MACHINE
    Software
        Microsoft
            Windows
                CurrentVersion
                    Control Panel
                        Extended Properties
                            {305CA226-D286-468e-B848-2B2E8E697B74} 2
                                "%SystemRoot%\System32\access.cpl"=[DWORD] 0x07
```

“{305CA226-D286-468e-B848-2B2E8E697B74} 2” is the string representation of the SCID_CONTROLPANELCATEGORY, which is composed of PSGUID_CONTROLPANEL and PID_CONTROLPANEL_CATEGORY (both defined in Shlguid.h).

The term “Extended Property” derives from the fact that you can access these properties from script through the **ExtendedProperty** method on the FolderItem object.

Sample Microsoft JScript® code to enumerate all the CPLs and their Category IDs

```
var strSCID = "{305CA226-D286-468e-B848-2B2E8E697B74} 2";
var shell= new ActiveXObject("Shell.Application");
var cpls = shell.Namespace(3).Items; //ID for Control Panel is 3

for (i = 0; i< cpls.Count; i++)
{
    var fldrItem = cpls.Item(i);
    Document.write( " The cpl " + fldrItem.Name +
                    " belongs to category id " +
                    fldrItem.ExtendedProperty(strSCID));
}
```

Allowed values of category IDs

0x00000000 : Other Control Panels. Any applet that does not specify a category ID ends in this bucket.
0x00000001 : Appearance and Themes.
0x00000002 : Printers and Other Hardware.
0x00000003 : Network and Internet Connections.
0x00000004 : Sounds, Speech, and Audio Devices.
0x00000005 : Performance and Maintenance.
0x00000006 : Date, Time, Language and Regional Options.
0x00000007 : Accessibility Options.
0x00000008 : Add or Remove Programs.
0x00000009 : User Accounts.
0xFFFFFFF : Do not put me in any category - Will not appear in the Control Panel categories.

For Control Panel items implemented as Shell namespace extensions (for example, Fonts or Scheduled Tasks), specify the Category ID in the registry under the CLSID entry.

EXAMPLE The registry entry for the Administrative Tools folder will be:

```
HKEY_CLASSES_ROOT
CLSID
{D20EA4E1-3957-11D2-A40B-0C5020524153}
" {305CA226-D286-468e-B848-2B2E8E697B74} 2"=[DWORD] 0x05
```

S2.6 Enable traversing Network Address Translation

Networking applications should work with Internet gateway devices (IGDs), which are often used in home networks. Networking applications that have Internet connectivity must not assume that they will have a direct connection to the Internet.

The proliferation of home networks has introduced Internet gateway devices (IGD) into the home. IGDs employ Network Address Translation (NAT) to provide multiple PCs on the home network the ability to share a single publicly routable IP address. NAT keeps a table of connection flows and will drop inbound connections (Internet to the home network) that do not map to an outbound flow.

To ensure end-to-end connectivity, applications should use Universal Plug and Play (UPnP) to detect the presence of an UPnP-compliant IGD. Upon detection of the UPnP-compliant IGD, the application should create port mappings in the NAT to allow inbound connectivity.

Applications must not assume they will be able to reserve specific ports as the port may already be reserved by another application. Use the **IStaticPortMappingCollection** API to obtain a list of current port mappings and add and remove port mappings as required.

A user should not have to manually enter port mappings in the NAT for an application to receive traffic from the Internet. Manual

configuration is unwelcome by customers and in some cases is not possible due to the technical nature of the task.

An application should remove all port mappings at the conclusion of session. Applications that run as a service and have a static port mapping may leave the port mapping to allow for Internet connectivity.

NOTE Gaming applications using DirectPlay in DirectX 8.1 will automatically receive this functionality and the above steps are not required for traversing UPnP IGDs.

Implementation Details – S2.7

For more information about UPnP APIs, see
<http://msdn.microsoft.com/library/>.

For more information about the Universal Plug and Play forum, see
<http://www.upnp.org/>.

For more information about NAT, see
<http://www.ietf.org/rfc/rfc1631.txt>.

S3.0 Optimized for Accessibility

Summary of Accessibility Guidelines

The Microsoft Designed for Windows—Optimized for Accessibility program offers you an accessibility blueprint that will increase your product's accessibility and help you reach new customers. Applications that support accessibility will meet the needs of a broad customer base, increase customer satisfaction, and also benefit mainstream users.

The Optimized for Accessibility guidelines will help developers of desktop applications, as well as developers of client components of Microsoft® .Net Server products, design and implement accessible products that leverage the Microsoft Windows® XP platform. However, additional accessibility considerations may pertain to your product. For example, specific regulatory and technology requirements exist for telecommunications products and Web products. Also, additional work may be required to ensure compatibility with assistive technologies.

Resources

Supporting information about these guidelines is available at
<http://www.microsoft.com/winlogo/software/default.htm>.

Information about U.S. Access Board Accessibility Guidelines & Standards can be found at <http://www.section508.gov>.

Accessibility guidelines for Web products can be found at <http://www.w3.org/WAI>.

For assistive technology compatibility needs, contact the assistive technology manufacturers. For an overview of assistive technology, see the Microsoft Accessibility Web site at <http://www.microsoft.com/enable/at/default.htm>.

Accessibility Guidelines List

- Support standard system size, color, font, and input settings
- Ensure compatibility with the High Contrast option
- Enable programmatic access to all user interface elements and text
- Provide keyboard access to all features
- Expose the location of the keyboard focus
- Provide user-selectable equivalents for non-text elements
- Do not rely exclusively on sound to convey information
- Avoid flashing elements
- Create accessible documentation about accessibility features

Accessibility Guidelines

S3.1 Support standard system size, color, font, and input settings

Overview

Your application should read, use, and preserve system-wide user interface (UI) settings when displaying customized controls or window content. The system-wide settings adopted by a user enhance your product's accessibility; your application should use them where possible, and not disable or disregard them. An example of a system-wide setting is a request for extra keyboard help.

This guideline benefits users who need to adjust system settings to avoid eye strain or to make text easier to read. It also helps individuals with visual or cognitive impairments who find certain font faces easier to read than others. Usability studies show that people using a mouse, especially those with dexterity or visual disabilities, are able to target large objects more easily and quickly than smaller ones. Therefore users can more efficiently use your application with the settings they have specifically chosen.

NOTE System-wide settings are set either programmatically or by the user through the Windows XP Accessibility Wizard and through Control Panel.

Requirements

Standard controls provided by User32.dll, Comctl32.dll, WinForms, and WebForms automatically support all of the required system-wide settings. Standard controls that are written according to the World Wide Web Consortium (W3C) HTML 4.01 and Cascading Style Sheets 2.0 (CSS2) specifications automatically support this requirement when executed by the Microsoft HTML parsing and rendering engine (MSHTML) and require no additional development work to satisfy this guideline.

Be careful when handling these settings in the following cases:

- Creating custom controls
- Creating owner-drawn controls
- Superclassing or subclassing to alter the normal standard control behavior
- Executing custom message handling before calling **DefWindowProc** (especially when drawing any non-client areas)
- Handling low-level input that bypasses normal mouse and keyboard messages (such as double-click and shift-state detection)

All applications should support system color settings for menus, dialog boxes, and other standard UI elements (see Guideline S3.2). Applications should also support system settings for the primary work area, mouse settings, and the use of sound. If you use customized controls, the following items should support the applicable system settings: text selection and editing, combo boxes, keyboard handlers, menus, scroll bars, ToolTips, status bars, and window frames.

Applications should not change, disrupt, or disable any UI settings. If your application encounters unrecognized settings, such as unknown flags or tags, your application should preserve—not remove or disable—these settings.

NOTE Many of the system settings are a part of an accessibility solution, but are not the complete solution in and of themselves. For example, if your application implements a custom cursor, the application code must pay attention to the system caret width as specified by the user. However, the custom cursor also requires programmatic access as described in S3.3.

Implementation Details—S3.1

To retrieve or set system-wide settings in Windows XP, call the **SystemParametersInfo** function and specify the appropriate system flag. For example, use SPI_GETKEYBOARDPREF or SPI_SETKEYBOARDPREF to retrieve or set the Keyboard Preference flag. **GetSysColor** and **GetSystemMetrics** are also used to retrieve

system settings. For a list of all system-wide settings pertaining to accessibility, see [Windows XP Systems Parameters](#).

S3.2 Ensure compatibility with the High Contrast option

Overview

Your application should read, support, and preserve the High Contrast option, which indicates that the user requires a high degree of contrast to improve screen legibility.

Typical users of the High Contrast option have visual impairments or may be subject to eye strain. Many require specific high-contrast combinations, such as white text on a black background. Reversing such combinations, such as by drawing black text on a white background, causes the background to bleed over the foreground and can make reading difficult, even painful, for some users. Hard-coded text colors can also make the content unreadable if the color is the same or very close to the background color.

Requirements

Standard controls provided by User32.dll, Comctl32.dll, WinForms, and WebForms automatically support High Contrast. Standard controls that are written according to the W3C HTML 4.01 and CSS2 specs automatically support this requirement when executed by the Microsoft HTML parsing and rendering engine (MSHTML) and require no additional development work to satisfy this guideline. Applications need to support the High Contrast option explicitly only when creating custom window classes or controls, or when altering the normal behavior of a standard window or control.

NOTE When High Contrast is enabled, Windows automatically switches the user to the Windows Classic Style. The High Contrast option can be used with any color scheme within the Windows Classic Style. Selecting a window and button style or color scheme through the Display section of Control Panel does not affect the High Contrast option setting.

You can determine the value of the High Contrast option using the **SystemParametersInfo** function with the SPI_GETHIGHCONTRAST constant. When this option is set, your application should do the following for any user interface (UI) elements you draw (the system menus and dialogs are automatically handled):

- Display all menus and dialog boxes using the color scheme returned by the **GetSysColor** function. This also applies to any other UI elements required to adjust colors in the application’s UI.

- Omit any images or patterns drawn behind text unless they are an essential part of the content, such as in a map program.
- Allow the user to adjust the colors used to display the contents in the application’s windows.
 - This should be possible using display options that override the colors normally used by the application or specified in the document, and should not alter the content of the document or affect other users.
 - The preferred method is to use the corresponding colors returned by the **GetSysColor** function, but the application can provide its own display options. If your application calls **GetSysColor** on any windows element, it should then use the returned RGB value for displaying that element. Your application must never update this system RGB value either by calling **SetSysColors** with a new RGB value, or by calculating with the returned RGB value and using the new calculated value for displaying the element.
 - Always draw foreground objects in colors designated as foreground colors and fill backgrounds with the corresponding background colors. This is required whether colors are selected using the **GetSysColor** function or through the application’s own options.

EXAMPLE Anything drawn using window text color (COLOR_WINDOWTEXT) should be drawn on the window background color (COLOR_WINDOW), and anything drawn using highlight text color (COLOR_HIGHLIGHTTEXT) should be drawn on highlight background (COLOR_HIGHLIGHT).

- Ensure that any information normally conveyed by color is available through other means, such as by sound or by visual display (such as patterning or cross hatching), because the user may have chosen a monochromatic appearance scheme.

NOTE Users can adjust the High Contrast option from the Accessibility Options section of Control Panel. The High Contrast color scheme differs from the High Contrast Mode. The High Contrast color scheme changes the system colors to colors that have obvious contrast; you switch to this color scheme by using the Display Options in Control Panel. The High Contrast Mode, which uses **SPI_GETHIGHCONTRAST** and **SPI_SETHIGHCONTRAST**, advises applications to modify their appearance for visually-impaired users. It involves such things as audible warning to users and customized color scheme (using the Accessibility Options in Control Panel). This requirement applies to High Contrast Mode only. For more information, see the **SystemParameterInfo** entry in the MSDN Library.

WHEN DOES THIS APPLY?

The High Contrast requirements do not apply to certain application features where the use of color is intrinsic and indispensable to the goal of the feature. Examples include:

- Palettes or swatches where the user selects from a range of displayed colors. In this case, the application can display the color but should provide a text description such as a name (light blue) or numeric value (RGB 0, 255, 255).
- Animation, video, and graphic images when the content is available through other means.

S3.3 Enable programmatic access to all user interface elements and text

Overview

Your application should provide programmatic access to user interface (UI) elements and text. Providing programmatic access to UI elements and text enables assistive technologies such as screen readers or speech recognition software to communicate information about the UI and text to users who need this information presented in an alternative format.

Microsoft Active Accessibility® is the recommended technology for providing programmatic access to UI elements and text. For more information, see the [Microsoft Active Accessibility 2.0 SDK](#).

Applications that provide programmatic access to UI elements and text are usable by a wide audience. Using Active Accessibility to meet this requirement will make it easy for assistive technology vendors to expose information about your application to assistive technology product users.

Requirements

Your application must provide programmatic access to the following UI elements and text:

- All UI elements—allows assistive technologies to identify and manipulate your application's UI elements.
- Descriptive titles on windows, frames, objects, and pages—allows people using assistive technologies, especially screen readers, to use the title to understand the context of the frame, object, or page in the navigation scheme.
- Alternative text—allows assistive technologies to provide text descriptions of non-textual UI elements, such as graphics. Many users turn off graphics, and screen readers cannot interpret these non-textual elements; therefore, descriptions for graphics are necessary.

- Text content—allows assistive technologies to access the content of an application, such as the text in a document, and describe it to the user.
- Data tables—allows assistive technologies to help users understand the information in a table.

For more information about meeting each of these requirements, see [How to Provide Programmatic Access to UI Elements and Text](#).

S3.4 Provide keyboard access to all features

Overview

Your application should provide keyboard access to all features so that a mouse or other pointing device is not required for its use.

Many users, with and without disabilities, prefer keyboard access. People who are blind or have mobility impairments that prevent the use of a mouse, power users, and people with repetitive stress injuries are typical users of keyboard access. For some users, keyboard access may be the only possible means of interacting with a computer. This is often the case for users of assistive technology products, even if the keyboard access is indirect, with the assistive technology product performing the keystrokes on behalf of the user.

Requirements

Standard controls provided by User32.dll, Comctl32.dll, and MSHTML.dll support all of the required keyboard behavior when given the correct labels and attributes.

You should explicitly provide keyboard support in the following cases:

- Creating custom window classes or controls.
- Altering the normal behavior of a standard window or control. The altered element should support keyboard behavior that is equivalent to the behavior of the standard control.
- Assigning keyboard navigation in windows or controls, such as dialog boxes.
- Using speech recognition technology for user input or commands.
- Creating controls using markup language.
- Using client-side scripting or plug-ins for user input or commands.

NOTE Keyboard behavior of standard window classes, including labels and attributes for common controls, is documented in [Guidelines for Keyboard User Interface Design](#).

EXAMPLE Implementation details for creating keyboard access for controls hosted by MSHTML.dll, as well as an example of creating dynamic HTML event handlers, can be found in [Making Web Pages More Accessible](#).

WHEN DOES THIS APPLY?

The keyboard access requirement does **not** apply to applications that rely on specialized input devices, such as graphing tablets. It also does **not** apply in situations where the mouse targets are no larger than a pixel—such as when painting with the mouse. These features may rely on the MouseKeys feature built into the Microsoft Windows 32-bit operating system that allows users to move the mouse pointer using the keyboard. However, this is not acceptable for drawing when the user can independently manipulate separate text and graphic objects. This exemption applies to individual features within a product, not to the entire application.

S3.5 Expose the location of the keyboard focus

Overview

Your application should visually indicate the location of the keyboard focus, and notify other software applications of this location by using Microsoft Active Accessibility or by moving the system caret. Standard controls provided by User32.dll and Comctl32.dll automatically expose the keyboard focus.

Assistive technology products use the location of the keyboard focus to determine which control's information is relevant. This information is used to describe the text or object to users who are blind. Panning software supported by many display adapters and assistive technology products, such as the Magnifier accessory included with Microsoft Windows XP, use the keyboard focus location to pan and enlarge that part of the screen for users who are visually impaired.

Requirements

The application's active window should display a visual focus indicator at all times so that users can anticipate the effects of their keystrokes.

Applications should programmatically expose the keyboard focus through Microsoft Active Accessibility. If using Active Accessibility is not feasible, the application can indicate the focus location by moving the system caret. The caret is normally the blinking vertical bar that the user sees when editing text, but it can be placed anywhere on the screen, made any shape or size, and even made invisible. If it is invisible, it can be moved to indicate the focus location to applications without disturbing what the user sees on the screen.

References

For information about making the focus visible, and about providing programmatic access through Microsoft Active Accessibility or the system caret, see [Guidelines for Keyboard User Interface Design](#).

WHEN DOES THIS APPLY?

The requirement to expose focus applies to any feature that provides keyboard access. The requirement does not apply to applications and features that are exempt from the keyboard access requirement as described earlier in this section.

S3.6 Provide user-selectable equivalents for non-text elements

Overview

Your application should provide a text equivalent for all non-text graphical and multimedia elements essential to an application’s user interface (UI) or content. These elements are commonly found in Web content and multimedia presentations. Non-text elements include graphics, image map regions, animations (such as animated .gif files), applets, sounds, video, and all sounds played with or without user interaction.

Users who are blind or visually impaired may not be able to see the essential non-text element and may need a textual description to effectively use the application. Users who are deaf may not be able to hear the essential non-text element and also may need an alternative, such as captioning, to effectively use the application.

Requirements

For each essential non-text element, provide a user-selectable equivalent alternative, such as alternative text for pictures or captioning for audio and video context. If your application is used to author non-text elements, it should provide a way for the author to create, edit, and reuse text equivalents.

Non-text elements are considered *essential* when they contain visual information, speech, or general audio information that the user needs to understand the content. If an application has major features that use graphics, audio, video, or animation, a user must be able to turn these activities on and off, and be able to choose an alternative way to access the information (for example, ALT attributes for pictures in HTML or static text for text in a marquee). For an audio or video context, the application should provide captions and audio descriptions.

For alternatives prepared for sound or video dialogs, support the Windows operating system ShowSounds option so that users can select captioning or another alternative format.

For captioning or audio descriptions, include the following features:

- User-selectable synchronized captioning to convey the narration or audio portion of multimedia presentations.
- The ability for the user to start, stop, and pause synchronized captions or audio description with no loss of information.
- The ability for users to turn off animations. All the information that is conveyed in an animation should also be conveyed when that animation is turned off.
- The ability for users to turn alternate presentations on and off.
- The ability for users to stop or pause an animation or audio track with no loss of information.

S3.7 Do not rely exclusively on sound to convey information

Overview

Your application should not convey important information exclusively by sound. If sound is the default method for conveying information, the application should provide other options to express this information. To provide alternatives to sound, applications can leverage the ShowSounds and SoundSentry Windows Accessibility options.

Users who are deaf or hard of hearing or users in a noisy environment will not be able to use the application if important information is conveyed with sound only. Users of an application running on a server that does not support audio services will benefit from having sound events conveyed in an alternative manner.

NOTE Users can modify their ShowSounds and SoundSentry settings through the Windows XP Accessibility Wizard and Control Panel. The operating system handles SoundSentry so the application is not required to check for that setting.

Requirements

Users select the ShowSounds option to notify applications that all information should be conveyed visually rather than by relying on audible means. You can determine the current value of the ShowSounds option with the **SystemParametersInfo** function to query SPI_GETSHOWSOUNDS. If the ShowSounds option is selected when the application starts, the application should either turn on these additional visuals or explicitly ask users if they want them.

If an application conveys information by audio and video content that is longer than several seconds, use closed captioning or synchronized text highlighting. For more information, see [Captions and Audio Descriptions for PC Multimedia](#).

Applications are not required to display visual equivalents of music or other sounds that convey no information. However, they are required to display a visual indication that such sounds are playing when the ShowSounds flag is set. When a sound merely emphasizes a visible event, the application does not need to provide an additional visual indication that the sound is present.

EXAMPLE An application has an option for a user to choose visual or sound feedback for event notifications. This application can set the option to display visual feedback by default when the ShowSounds option is active.

S3.8 Avoid flashing elements

Overview

Your application should avoid flashing content to eliminate the risk of causing seizures in some people using the product.

Flashing content can cause seizures in people with seizure disorders such as photosensitive epilepsy. Movement can also be distracting to some users, making it difficult for them to use the application effectively. Some assistive technology vendors direct their customers to set a particular blink rate to avoid conflicts with their screen sampling rates.

Requirements

Your application should not cause the screen to flicker, flash, or pulsate and should not contain flashing or blinking text, objects, or other elements. If your application must contain flashing or blinking elements, it should use the system caret blink rate to set the rate of blinking or flashing, or allow the user to turn off flashing entirely. This allows users to adjust the rate as required.

References

For more information about accessing the system caret blink rate, see [Flashing User Interface](#) and the [GetCaretBlinkTime Function](#).

S3.9 Create accessible documentation about accessibility features

Overview

Your application’s documentation should be accessible to all users. Ensure that the documentation fully documents the accessibility features of your product and that the documentation itself is accessible.

Many users have difficulty reading or handling conventionally printed material or using online documentation, such as assistive technology product users, keyboard users, and people with cognitive disabilities. Documentation that is accessible enables users who need to use the accessibility features to discover information about these features.

Requirements

Document all product features and options that benefit accessibility, such as options that enable users to customize font size, color, sound, and item size. In addition document access keys, keyboard shortcuts, complex keyboard procedures, and controls that do not conform to normal conventions. If your product is used to create content, such as Web content, your product documentation should show users how to create accessible content.

Provide your product documentation in a variety of accessible formats such as print, online, multimedia, HTML Help, or CD-ROM tutorials. Printed documentation shipped in the box should also be provided in electronic form on CD or on the Web.

For electronic or online formats, the documentation should meet the applicable Designed for Windows—Optimized for Accessibility guidelines:

- Support standard system size, color, font, and input settings
- Ensure compatibility with the High Contrast option
- Enable programmatic access to all user interface elements and text
- Provide keyboard access to all features
- Expose the location of the keyboard focus
- Provide user-selectable equivalents for non-text elements
- Do not rely exclusively on sound to convey information
- Avoid flashing elements

An effective way to provide an alternative format is to give users the option to download files onto their systems. The content of downloadable files must match the online documentation. Accessible file formats include plain text files (.txt), Word files (.doc), and compiled HTML files (.chm).

S4.0 Optimized for .NET

Summary of the Optimized for .NET Guidelines

The requirements for an application to be considered "Optimized for .NET" vary according to the functionality of the application.

Applications that do not interact with the Internet will not meet these qualifications. But for Internet applications that benefit from peer-to-peer communication functionality, and/or those which rely on Authenticated Programmable Web Services, there are requirements to be considered "Optimized for .NET".

In some instances, a single application may have both attributes, in which case your application must meet all of the criteria in this section.

For Peer-to-Peer enhanced applications, the following are required:

- Support for Windows Messenger on the Client
 - Logging on
 - List of online/offline contacts
 - Ability to instant message and use voice and video

For Passport Authenticated Web Service Resource Applications, the following are required:

- Support of Passport on the Client and on the Server
- Use Passport in WebService Calls

Optimized for .NET

S4.1	Communication Enhanced Peer-to-Peer Applications
S4.2	Passport Authenticated Web Service Resource Applications

Optimized for .NET Guide

S4.1 Communication Enhanced Peer-to-Peer Applications

S4.1.1 Support for Windows Messenger on the Client

In Windows XP, the ability to communicate with friends, family, and co-workers, becomes an integral part of the .Net experience. For this reason, client applications can take advantage of the ability of Windows Messenger to enable this kind of real-time communication.

Windows XP enables your application to sign the user into .NET Messenger Service either manually or using **AutoSignIn** (if not

already signed in) and iterate and manipulate the collection of contacts using the MessengerUA object. This collection includes information about online status, email address and familiar name. Furthermore, the identity of the user can be retrieved from these API's

When showing the list of buddies, the application should use the familiar messenger icons and the familiar tree view. The *only* exception is if using this User Interface significantly alters the application design.

S4.1.2 Integrate Contacts and Online Status

Your application should display a user's contact list and the corresponding status of those contacts. The following API's (and others exposed in the Messenger SDK) should be implemented in the application for a seamless communication experience for users. The collection of contacts is fully programmable using the built in functions in the **MessengerUIAutomation** library. The **AddContact**, **FindContact**, **GetContact** methods provide you with simplified and write access to the collection.

Furthermore, common messenger functions are available through this library. These include **AutoSignIn** (to automatically sign the user into Messenger), **InstantMessage** (to display the instant message UI and send a "Toast" popup to a contact to initiate a chat session), **SendFile** (to initiate the streaming of a file over the net), and **StartVoice** (to start a voice conversation).

A great number of peer-to-peer scenarios become possible when the online status of a user's contacts, and the ability to share information with them is available.

Implementation Details

All of the information relating to using the MessengerUA object and all of the features discussed above can be found in the Messenger SDK. If you need access to the Messenger SDK please look on <http://www.microsoft.com/winlogo> or email swlogo@microsoft.com

S4.2 Passport Authenticated Web Service Resource Applications

S4.2.1 Support of Passport on Client and Server Applications

In order to leverage Microsoft .NET Passport in your client application, the web service from which the online data is being retrieved, must also implement passport. Utilizing .NET resources in your application include scenarios where you retrieve information or files based upon a user's identity. For example, if a user opens a stock ticker application,

passport can be used to identify that user and download the stocks that are of interest to him.

For information on implementing Microsoft .NET Passport on your web server visit <http://www.passport.com>.

S4.2.2 Use Passport Credentials in WebService Calls

When calling a web service, in order to identify the user, you will want to use the PUID (Passport User ID). In order to obtain this unique identity, you must initially query a Passport web page that challenges for Passport credentials.

Implementation Details

The support for Passport 1.4 is implemented in the Wininet.dll that ships within Windows XP. When a Windows XP client is challenged for Passport credentials, the Windows XP operating system takes over via Wininet.dll on behalf of the user either providing the credentials if they are available, or prompting the user for the credentials.

S5.0 Optimized for Enterprises

Summary of the Optimized for Enterprises Guidelines

Overview

The following guidelines define what an application must do to meet the "Optimized for Enterprises" standard. These guidelines include additional requirements that supplement the core Designed for Windows XP requirements and help address the some of the needs of businesses:

Reliability:

- Enhanced Reliability
- Execute while a virus scanner is running
- Degrade gracefully when services are unavailable

Deployability:

- Install using a Windows Installer-based package that passes validation testing
- Execute appropriately in a multi-lingual environment

Manageability:

- Files outside of your application folder have associated file-types

Security:

- GINAs must support Smart Card login
- Support Secure Credential Management
- Run in a highly secure configuration
- Do not make insecure additions to a secure desktop
- Use of network connections must be secure
- All executable components must be signed

Optimized for Enterprises List

S5.1	Enhanced Reliability
S5.2	Execute while a virus scanner is running
S5.3	Degrade gracefully when services are unavailable
S5.4	Install using a Windows Installer-based package that passes validation testing
S5.5	Execute appropriately in a multi-lingual environment
S5.6	Files outside of your application folder have associated file-types
S5.7	GINAs must support Smart Card login
S5.8	Support Secure Credential Management
S5.9	Run in a highly secure configuration
S5.10	Do not make insecure additions to the secure desktop
S5.11	Use of network connections must be secure

S5.12 All executable components must be signed

Optimized for Enterprises Guide

S5.1 Enhanced Reliability

There are three commonly misused resources that can be easily tested: heap, critical sections, and handles. In each case, the misuse can result in less reliable applications and failures with subtle circumstances that impact customers but may not be easily reproduced. These failures can include crashing applications that loose data.

Your application must be able to perform all of its primary *and* secondary functionality with no errors, Dr Watson reports or events that launch the debugger while running under AppVerifier configured to detect heap corruptions, invalid locks usage (critical section use), and invalid handle usage. For more information, search for "AppVerifier" in the MSDN library:

<http://msdn.microsoft.com/library/default.asp>. For information on using AppVerifier to test for this requirement, see the *Optimized for Enterprise Application Test Framework* which is available on <http://www.microsoft.com/winlogo/downloads/software.asp>.

NOTE It is recommended that samples, and other supplemental extras included with your application also pass this testing, but it is not part of the requirement.

The following is more detail on the issues AppVerifier detects when configured to trap heap corruptions, invalid locks usage (critical section use), and invalid handle usage.

Heap use

Dynamic memory allocations come from the Heap. Heap errors can result in subtle errors in an application, security holes, and can cause an application to crash. There are several invalid uses of the heap, including:

- Allocating memory, but writing beyond the end of the allocation (buffer overruns)
- Use of allocated memory after it was freed
- Freeing an allocation twice
- Freeing unallocated memory
- Using wrong heap pointers

Critical section use (Locks usage checking)

Critical sections are user mode synchronization primitives used to guarantee exclusive access to application data in a multi-threaded environment. Some invalid use of critical sections includes:

- Releasing a critical section that is not owned by the current thread
- Terminating threads while they own critical sections
- Use of a critical section before being initialized
- Leaked critical sections (i.e. **DeleteCriticalSection** not called)
- Double initialized critical sections

Handle use

Kernel handles which include handles to files, events, etc. can also be misused:

- Reuse of a handle after being closed
- Use of a handle for an operation requiring another handle type (you cannot read from an event)
- Use of a random handle value
- Use of a null handle or a pseudo-handle (for example, values returned by **GetCurrentProcess()** when not permitted).

To see why these kinds of errors can have bad consequences, let's look more closely at reuse of a handle after being closed. When a handle gets closed, the value previously assigned to it will be reused by the system. Let's say that you have a file handle open and you close it, but you keep the value of the handle in some global variable. Now if some other part of the process opens a file handle for a totally different reason (might not even be your code), the new handle might have the same value. If you still hold on to the old value in some variable and continue to use it, you will write in the wrong file.

S5.2 Execute while a virus scanner is running

Many device conflicts and application stability issues are exposed when the application runs on a system with kernel mode active virus scanning. Applications must be able to perform all primary functionality while a virus scanner's kernel mode service is actively scanning for viruses. The reverse is also true: the kernel mode active virus scanning service must not be disrupted when applications perform their primary functionality. Applications should also perform correctly while a user mode virus scanning utility is used to scan one or more files, but that is not part of this requirement.

Any application interactions with an active virus scanner service must be appropriate and must be handled gracefully. If active virus scanning has any effect on an application's functionality, the application must ensure that the benefits of the interaction and the options for the user are both clear.

WHEN DOES THIS APPLY?

This requirement applies as described above for all applications other than virus scanners. A virus scanner must be tested to provide some assurance that it does not conflict with other file handling applications and drivers.

If your application is a virus scanner, its runtime must operate correctly on a system with an active file management service running, and while a backup application is performing a backup and restore of several folders.

S5.3 Degrade gracefully when services are unavailable

If an application depends on a service that has been disabled or not installed (perhaps to eliminate a security issue), it must not crash, hang, display cryptic error messages, or simply fail to operate. Instead it must inform the user, giving guidance for resolving the issue.

If the application depends on a service that's not installed, it must display an error message identifying the service that is not installed. The application must be able to do this when executed, but should also be able to inform the user during installation.

If the application depends on a service that's installed but either not started or disabled, then during installation and also when executed, the application must do one of:

- Inform the user about the disabled service and offer to reconfigure the service.
- Inform the user about the disabled service and how to enable it in the services UI.
- Configure the dependent service to start appropriately without informing the user. The application should only start a service if the service's startup type is "manual". The application must not automatically enable disabled services.

NOTE Enabling or installing services requires administrative or LocalSystem privilege. A limited user and applications running as user will be able to start a service with startup type is set to "manual" but not a "disabled" service.

WHEN DOES THIS APPLY?

If the *Windows Installer* service or *Windows Management Instrumentation* service is stopped or disabled, it is acceptable for applications to not install and for features to not function without providing any user guidance on how to resolve the issue. However, the application must not crash, hang, or display cryptic error messages..

S5.4 Install using a Windows Installer-based package that passes validation testing

Your application must install itself using the Windows Installer service. To do this, your install must be in the form of a Windows Installer-based package. You must validate that the package is properly constructed by:

- Running an Internal Consistency Evaluation tool (such as msival2.exe, available in the Windows Installer SDK) in conjunction with the suite of Internal Consistency Evaluators (ICEs) contained in the file named Logo.cub. These are available from the Windows Installer SDK at <http://www.microsoft.com/msdownload/platformsdk/sdkupdate/>. You can also use the Logo.cub file in conjunction with ICE tools provided by leading install tool vendors. Warnings given by the ICE tools are not fail conditions and should be addressed in the next release. All errors identified by the ICE tools must be fixed.
- Testing that all Windows Installer-based packages of your core application are advertisable in a manner that operating system entry points, such as shortcut activation and file activation, can trigger install on demand.

Leading vendors of install tools have versions of their tools to enable easy authoring of Windows Installer-based packages. Contact your tools vendor for more details. You can also create your own Windows Installer-based package. The Microsoft Platform SDK provides a detailed example for implementing a Windows Installer-based package.

Advertising

In order to support advertisement at the operating system level, the following tables in the Windows Installer-based package must be populated with advertising data: *shortcut*, *extension*, *icon*, and *Verb*.

It is recommended that you also populate *class*, *MIME*, *ProgID*, and *TypeLib*.

NOTE This does not mean your application must support feature-level advertising, (e.g. enabling the spell-checker to install on demand). The intent of this requirement is to ensure that your overall application can be installed on demand using the IntelliMirror features of Windows. The MSDN library contains more information on “IntelliMirror” and “install on demand” <http://msdn.microsoft.com/library/default.asp>.

WHEN DOES THIS APPLY?

If your application does not create file associations, it does not have to advertise via file extension. If your application does not add shortcuts, it does not have to advertise via shortcut. You still must support advertising such that users can install via the Add or Remove Control Panel item.

S5.5 Execute appropriately in a multi-lingual environment

Applications must be able to perform all primary functionality on systems that have printer names and paths that include non-English Unicode scripts, and using non-English user names.

Ensure that you test your application with paths that contain a mix of characters covering different Unicode scripts in all of the situations below:

- The user name contains Unicode characters. This will result in the path for the temp folder and the user's my documents folders to contain Unicode characters. It is acceptable for some supplemental features to fail gracefully in this case as long as there is a reasonable workaround to ensure it will not affect primary functionality. You must test with both an English user name, and with a user name in a non-European language such as Japanese. A well written Unicode based application will pass this test without needing any changes to system settings. However, it is currently acceptable for applications that are not fully Unicode to be tested on a system where the "Language for non-Unicode programs" is set to match the language of the user name. This setting is made on the advanced tab of the regional and language options in the control panel.
- If your application opens or saves files, your application must be able to open and save files on a folder with a mix of characters including European and non-European Unicode scripts.
- If your application provides a UI to allow printing, then your application must be able to print to a printer with a name that includes a mix of characters from European and non-European Unicode scripts.

NOTE These last 2 bullets are different from core requirement 1.1 which states that applications must not crash in these situations. In this requirement, the application must be able to use the path and printer names, not just remain stable.

S5.6 Files outside of your application folder have associated file-types

Every non-hidden file (files that do not have the "hidden" file attribute set) that your application creates outside its directory in "Program Files" (see requirement 2.5) must have an associated registered file-type. This includes:

- Files created during installation
- Implementation and data files
- User created files that are native to your application

WHEN DOES THIS APPLY?

Files that the application creates in a temporary folder such as the user's "Temp" folder do not have to have file associations.

If the file-type is already registered, no action is required, though you may take over the file-type if you wish.

If the file-type is not already registered, you must do the following for each new file type:

- Provide an icon so that none of the files that your application creates is identified by the default Windows icon.
- Provide a friendly type description, for example, "Outlook Offline Mail File."
- Ensure that each file-type has an associated action when double-clicked (e.g. launch your application and load the file), or is designated as "NoOpen".

The **NoOpen** designation may be used for files that you don't want users to open. When the user double-clicks a file marked as **NoOpen**, the operating system will automatically provide a message informing the user that the file should not be opened. Note that if an action is later associated with a **NoOpen** file type, the **NoOpen** designation will be ignored.

WHEN DOES THIS APPLY?

If your application saves or exports file types that aren't native to your application, or are commonly identified as native to other applications, you are not required to register the file type. Your application may save the file, even though the file will have a default Windows icon and no default actions.

Individual files that are placed in appropriate locations outside your application's folder and are not meant to be seen or opened by the user, may be marked with the hidden file attribute, and do not need to have registered file association. See chapters 2 & 3 of the core requirements for more information on appropriate locations for files.

S5.7 GINAs must support Smart Card login

An incorrectly written Graphical Identification and Authentication dynamic-link library (GINA) can break Smart Card authentication on the PC.

If you install a GINA, it must work correctly with smart card authentication on the system and must also allow smart card authentication to a terminal service. This is in addition to the GINA requirements in section 1.5.

S5.8 Support Secure Credential Management

Secure credential management includes appropriate prompting for credentials and storing credentials.

- Support for User Principal Name (UPN) format:
username@domain.

A user can chose to login using either the User Principal Name format which includes the UPN suffix, or just the User Name without the UPN

suffix, by simply typing their name in either one in the login dialog. Both are valid login forms. An application must be able to perform all primary functionality when a user uses the User Principal Name format.

The following requirements only apply to applications seeking the logo after July 31, 2002

- Accept smart card/PIN when appropriate, such as with the secure network communications protocols. Examples of such protocols include: the Kerberos protocol and Secure Sockets Layer (SSL).
- If the application will provide the option to securely save credentials for single-sign-on, it must do so in a secure fashion. If that option is employed, the application must provide seamless access on future attempts (single-sign-on).
- Only the one user associated with those credentials can use them. Any other user using your application must be prompted for their own credentials.

NOTE The Credential Management UI which works on top of the "Stored Usernames and Passwords" system provides this for you. It also accepts X.509 Certificates. The "Stored Usernames and Passwords" system works with protocols: Kerberos, Windows NT LAN Manager (NTLM), and SSL.

S5.9 Run in a highly secure configuration

Applications must be able to perform all primary functions in a highly secure configuration. In a highly secure configuration applications are restricted from using the unsafe communication protocol NTLM, strong authentication and account policies are set, group membership is restricted, etc. Applications should be written to not require unnecessary access so as to allow administrators freedom to restrict user access.

The specific definition of a *highly secure configuration* is a system with the predefined security template *Hisecws.inf* applied. This security template is included with Windows XP.

For more information, look up "Security Templates" in Windows XP "Help and Support" on the start menu, or use the following URL to bring up the Help and Support article:

hcp:ms-its:SCEconcepts.chm::/sag_SCEwhatis.htm. You can also view the settings contained in *Hisecws.inf* by opening it in Notepad.

NOTE Applying *Hisecws.inf* enforces secure account use and authentication. When testing using *Hisecws.inf*, you should either ensure that you are already following highly secure policies for passwords and accounts, or do not log off until restoring the system's original settings by applying *setup Security.inf*.

S5.10 Do not make insecure additions to the secure desktop

The secure desktop is a tightly controller UI presented in certain situations:

- The change password dialog.
- The locked computer screen.
- The UI presented when no user is logged on.
- The Security dialog on a domain member computer when the CTRL+ALT+DEL keys are pressed.

Most applications make no changes to the secure desktop. The standard UI on the secure desktop can only be modified when adding to or replacing the GINA, but it is possible for services to add dialogs or Windows to the secure desktop.

Because the secure desktop is running in a high access context, it is very important not to leave any security openings:

- Do not add items that run in a user context to the secure desktop.
- UI on the secure desktop must be securely bounded. Do not add HTML help or other potentially open ended UI elements. It must be possible to test every potential UI choice and ensure that no unintended access is available.

S5.11 Use of network connections must be secure

Applications using network connections must take common security precautions:

- When connecting via a Security Support Provider Interface (SSPI), applications must not default to use the Windows NT LAN Manager version 1 (NTLM v1) protocol.
- Sensitive data sent over the network must not be sent in clear text. Instead, data should be protected by an encrypted data stream, for example: `RPC_C_AUTHN_LEVEL_PKT_PRIVACY`, `ASC_REQ_CONFIDENTIALITY`, or `SSL`. Sensitive data includes password, user name, address, credit cards, or any other personal user information.
- Your application must be able to perform all primary functionality when NetBIOS over TCP is disabled.

S5.12 All executable components must be signed

For all applications signing up for the Designed for Windows logo program after July 31, 2002, all executable components in the application must be signed with an Authenticode certificate including a timestamp. All files included in your application with the following file types must be signed:

- All scripts handled by the windows scripting host (the file types are: VBS, VBE, JS, JSE, WSF, and WSH).
- EXE files
- Windows Installer packages (MSI)

You may go to the Certification Authority (CA) of your choice to get the certificates for these signatures, and you do not go to Microsoft for this. Since you will be required to have a VeriSign ID in order to submit for the Designed for Windows logo, this would be a natural choice for CA. Signing components is very easy, and information about it is available on <http://www.microsoft.com/winlogo/downloads/default.asp>. Look at the presentation on the "VeriSign Signing Process".

A prime reason for signing components is that an administrator can use Software Restriction Policies to prevent many types of unsigned executable files like script files (such as VBS) or even Microsoft Installer (MSI) files from executing. For more information, look up "Software Restriction Policies" in Windows XP Help and Support, or use the following URL to bring up the Help and Support article: hcp:ms-its:safeconcepts.chm::/Safer_topnode.htm

Another benefit of signed files is that users could examine any executable and have more information to evaluate if it is safe to run. The user would do this from within Windows Explorer by right-clicking on the file, selecting Properties, clicking on the Digital Signatures tab (if there is no such tab, then the executable has not been signed), selecting the signature, and looking at the details.

The signature prevents the file from pretending to be something it is not. For example, if a file is changed or replaced, the signature would not match and the substitution would be plain. Also, the signature can reflect new information. For example, if a signed application later was discovered to be a virus, the CA could revoke the certificate and all subsequent users would be able check and see that it would not be safe to run the application.

Section 3: Future Requirements

F1.0 Future Requirements

The guidelines outlined in this section are not required to comply with the “Designed for Microsoft Windows XP” Application Specification and are not required for the “Designed for Windows XP” logo, but are likely to be adopted as requirements for future logo programs.

Summary of Future Requirements

Microsoft recommends complying with these requirements because they will result in a better user experience for your customers, in addition to your application having an easier time meeting the requirements of future logo programs.

Many of the guidelines in Section 2 “Designed for Windows XP - Optimized” are also likely to become “Designed for Windows” logo requirements in the future.

Additional information about these guidelines may be available at <http://www.microsoft.com/winlogo/software/swfuture.asp>.

Future Requirements List

- F1.1 Install shortcuts correctly
- F1.2 Deliver meaningful user notifications
- F1.3 Support Long Path Names
- F1.4 Operate correctly on high-density displays
- F1.5 Use Windows Installer
- F1.6 Use SHGetFolderPath to determine special folder paths
- F1.7 Implement IPv6 protocol support
- F1.8 Networking applications and protocol support
- F1.9 Create isolated applications using side-by-side assemblies
- F1.10 Do not use NULL DACLs
- F1.11 Do not run as LocalSystem
- F1.12 Use Appropriate Sensitive Data Storage

Future Requirements

F1.1 Install shortcuts correctly

The Start menu is designed to give users easy access to launch applications. Usability studies show that when the Start menu and/or Windows Desktop become too cluttered, users have a very difficult time finding and launching their programs. This leads to a bad user experience.

- Do not place shortcuts on the Desktop except by request of the user. If an option to add such a shortcut is offered to the user, then this should not be checked by default. Automatic installs should not place any shortcuts on the Desktop.
- Game applications should place one shortcut for the main executable of the game in the Games folder. This makes the game readily accessible to the user. Game executables should not be put onto the Start menu.
- If any additional shortcuts beyond the main executable are required to be on the Start menu, then these should be placed in a subfolder using a *<publisher\product>* hierarchy.

That is, there would be only one entry on the Start menu for the publisher and under that entry would be folders for each product. Place any required ancillary functionality in this product folder, including shortcuts to: the main executable, the Readme file, web site access, update mechanism, uninstall, and so on.

- Although many items may be placed in the product folder on the Start menu as described earlier, the best practice is to avoid putting most items on the Start menu:
- Do *not* place shortcuts to documents, such as Readme files, in the Start menu.
If you have important information that the user should see, consider displaying that information during the install process or on the help menu of the application.
- Do *not* put shortcuts to help files in the Start menu. Users can access Help after they launch the application.
- Do *not* place shortcuts to remove the application in the Start menu. It is not needed because your application's uninstaller is in the Add or Remove Programs Control Panel item.

F1.2 Deliver meaningful user notifications

Since the release of Windows 95, software developers have made increasing use of the notification area (system tray) for varying purposes. Numerous icons and differing icon UI has resulted in the average user ignoring this area altogether. The increased clutter makes the computer harder to use and lessens the meaningfulness of legitimate notifications.

Observe the following guidelines for the notification area:

- Do *not* display general status information for services or background applications. Instead, run programs in the background, as necessary. Users can view service status or configure a setting via a second application or a control panel that provides a UI for your service. Users must explicitly launch this application from an icon in the Start menu.

- Do *not* use the area to change settings or other properties. Instead, provide UI for changing settings or other properties from within the application or from a Control Panel item.
- Do *not* use the area to provide an application's launch point. Instead, use only the Start Menu to launch your program or start your service. If your program or service launches automatically at startup, do not default to placing an icon in the notification area.

General guidelines

This section provides general guidelines that apply to all Windows XP notifications. Every notification should do the following, except where noted:

- Place an icon in the notification area.
- Include a balloon notification to provide summary information when the notification first appears. The title should be a short phrase that gives a high-level description of the event, e.g. "Now connected." The body of the balloon notification should provide more detail, e.g. "Now connected to MSN Internet Access."
- Do not use balloon notifications more than once per user session. For many notification events, it's appropriate to leave the icon in the notification area after the balloon has timed out or has been dismissed. All warning and critical-level notifications should keep their icon in the tray until the user has remedied the situation.
- Provide detailed information when the user clicks the balloon notification or icon. Simple informational events may not require additional information.
- Leave the icon in the notification area as long as it remains reasonably meaningful to the user. Then remove it.
- Open applications should deliver notifications within the context of the application; instead of using the notification area, issue notifications within the window frame.

Providing more information

Usually, notifications consist of more information than can fit in a balloon notification. If this is the case, then clicking the balloon notification or icon should display a "notification window"—that is, any window that gives the user more information about the event. From this window, a user can drill down to more detailed information as necessary.

The notification window should include a way to access a configuration dialog or page that allows the user to specify which events should post notifications. The configuration page should also include an easy way to disable any or all notifications provided by the service.

Sometimes it is necessary to notify users of many similar events at one time. If possible, your service should try to consolidate these events into one notification.

EXAMPLE If your service notifies the user of bills that are coming due in the next week, your service should display a single balloon notification that says "You have 4 bills due in the next week," rather than a separate balloon for each bill. The notification window should provide a way for the user to see or access all similar notifications—in this example, each overdue bill.

Levels of notifications

There are three general levels of severity for notifications: informational, warning, and critical. Critical notifications should be delivered only when data loss is imminent.

Providing too much information

It is easy when creating a notification delivery service to think that this particular notification is superior and ought to receive special attention. **Do not give into this temptation!**

Microsoft research shows that users perceive frequent or unsolicited delivery of notifications to be annoying. Furthermore, studies show that it is easy for users in such cases to lose trust in both the specific offending notification and in the notification method or UI (in this case, balloon notifications).

User presence awareness

Windows is aware when the user is not using the computer. In such cases, it will queue balloon notifications in a first-in, first-out order. The first balloon notification will remain displayed until the user returns to the computer, regardless of what the timeout value is. This is to ensure that the user will receive your service's notification.

Because of this, you can be assured that your notification will be delivered to the user. Again, do not deliver notifications more frequently than is prescribed in guidelines specific to notification severity.

Revoking and updating notifications

The user-presence awareness technology places responsibility on the notification provider to revoke out-dated balloon notifications. Sometimes notifications become out of date because more recent information is available. When this is the case, simply update the text of the balloon notification.

Your service should revoke notifications that no longer make sense. For example, if your service has recently delivered a single-event notification, "Your phone bill is due next week," and you want to deliver a similar notification, "Your insurance bill is due next week," your service should revoke the first balloon and deliver a multiple-event notification, "You have two bills due next week." Removal of

notifications involves removing the balloon notification and notification icon.

Remember, the timeout period only includes time when the user is present at their computer. Therefore, when a time-out occurs, assume that the user has read your notification. It is not necessary to repeat notifications more than once per user session.

F1.3 Support Long Path Names

Windows supports long file and printer names that may contain any UNICODE characters (with the exception of reserved file system characters, such as / ? * : etc). If the application exposes filenames to users, allows users to enter file names, or both, the application should support all valid Win32 file names, including Long File Names (LFNs) and UNC names.

EXAMPLE The application should properly recognize and display the following paths:

```
C:\documents and settings\joe user\my documents\my letter.txt
C:\documents and settings\Joe [joeuser]\my documents\Ca$h;flow\my
letter to André.txt
D:\Our folder\project 10\project 11\project 12\project 13\project
14\project 15\ project 16\project 17\project 18\project 19\project
20\project 21\ project 22\project 23\project 24\project 25\project
26\project 27\ project 28\project 29\project 30\filename.ext
\\server\sharename\joe user\my letter.txt
```

All Win32 functions that create, open, locate, and save files and folders use the constant MAX_PATH as the maximum buffer size for path information. The application should allow users to create files with a total path length up to MAX_PATH, and the application should open any supported files the user creates with the application and any other application on paths up to MAX_PATH long.

The MAX_PATH constant is defined in the Platform SDK support file Windef.h. In the current platform SDK, the value of MAX_PATH is “260.” Use the MAX_PATH constant name instead of coding the value into the application. Using the constant instead of the value will help you quickly adapt the application to future versions of Windows, which may support longer paths but use the same constant name.

For more information about supporting valid Win32 long file names, see the Platform SDK and the topic “File Name Conventions” in the MSDN library: <http://msdn.microsoft.com/library/default.asp>.

Note on DBCS applications

While many languages can be supported with double-byte character sets, UNICODE is strongly recommended - particularly for international versions. In the future, UNICODE support will become a requirement, and applications should support the full UNICODE character set for file names. Applications that are submitted for any double-byte language should be tested for compliance with the long

path names requirement by using LFNs based on those character sets.

Support printers with long names and UNC paths

Windows allows names for printers up to 220 characters long. The application should start and function without errors, even if the Windows default printer has a long name. If the application supports printing, it should accept and display printer names up to 220 characters long (if any printers are displayed by name in the print dialogs), and print to devices with long names.

The following are examples of long printer names:

```
\PrintServer44.domain.com\Duplex printer: number 0047 (accounting group)  
Color laser - Research & Design Dept IP 123.456.78.9 see Fred or Wilma
```

Note that commas and exclamation points are not legal printer name characters.

F1.4 Operate correctly on high-density displays

With 133 dots-per-inch (dpi) displays on the market and 200 dpi displays coming soon, it is important to test applications with large fonts and icons. In Windows XP, the maximum dpi setting is now 480 dpi, which enables text that is more legible and of much higher quality than an equivalent 96-dpi monitor. Some items that use this font feature are dialog boxes, buttons, title bars, and input by pen and speech.

All dialog boxes, controls, text, and so forth in the application should be legible and useable when the application is running on a 133-dpi display.

F1.5 Use Windows Installer

The application should install itself by using the Windows Installer service. For information on how to best implement this, see the documentation on Windows Installer. This is under Setup in the Management Services section of the Platform SDK documentation on <http://msdn.microsoft.com/library/default.asp>. See also Chapter 2 of the "*Certified for Windows*" Desktop Specification at <http://msdn.microsoft.com/certification/download.asp#english>.

Most third party setup authoring tool vendors have versions based on Windows Installer and make it easy to meet this requirement.

F1.6 Use SHGetFolderPath to determine special folder paths

SHGetFolderPath is the preferred method of retrieving paths for specific situations in several of the requirements. There are many other special folders that you can access from **SHGetFolderPath**.

Whenever you access any of the special folders (see the implementation details), the application should use the Win32 APIs to

dynamically obtain the proper language-specific folder names. The preferred way to do this is using the **SHGetFolderPath** API with the appropriate CSIDL constant. This function behaves consistently across Windows 95, Windows 98, Windows Me, Windows NT 4.0, Windows 2000, and Windows XP.

Implementation Details – S4.6

This API is redistributable through the Shfolder.dll. Software vendors are encouraged to redistribute this component as much as possible to enable this support on versions of Windows operating systems earlier than Windows XP. Windows 2000 includes this DLL as a protected system file and, as such, this DLL cannot be replaced on Windows 2000 or later.

To help ensure that the application can run on Windows 95/98, Windows Me, Windows NT 4.0, Windows 2000, and Windows XP, always link to the **SHGetFolderPath** implementation in Shfolder.dll. Windows 2000 and Windows XP natively implement **SHGetFolderPath** in Shell32.dll, but other versions of Windows do not include **SHGetFolderPath** in Shell32.dll.

For full descriptions of all CLSID values, see the article titled "CSIDL Values" in the MSDN Library. You can find it and other references by going to <http://msdn.microsoft.com/library/default.asp> and doing a search for "CSIDL Values". The following is a list of several significant CLSID values.

Standard folder	CSIDL constant name
Documents folder ([user] profile)	CSIDL_MYDOCUMENTS
Music folder ([user] profile)	CSIDL_MYMUSIC
Video folder ([user] profile)	CSIDL_MYVIDEO
Desktop folder ([user] profile)	CSIDL_DESKTOPDIRECTORY
Start menu ([user] profile)	CSIDL_STARTMENU
Programs folder (under Start menu in [user] profile)	CSIDL_PROGRAMS
Application Data ([user] profile)	CSIDL_APPDATA
Documents folder (All Users Profile)	CSIDL_COMMON_DOCUMENTS
Music folder (All Users Profile)	CSIDL_COMMON_MUSIC
Video folder (All Users Profile)	CSIDL_COMMON_VIDEO
Start menu (All Users profile)	CSIDL_COMMON_STARTMENU
Programs folder (under Start menu in All Users profile)	CSIDL_COMMON_PROGRAMS
Application Data (All Users Profile)	CSIDL_COMMON_APPDATA
Local (non-roaming) data repository for apps	CSIDL_LOCAL_APPDATA

F1.7 Implement IPv6 protocol support

Concerns with the impending shortage of Internet Protocol (IP) Version 4 (IPv4) addresses and the desire to provide additional functionality for modern devices has led to the development of IP Version 6 (IPv6).

To handle this, new applications should take advantage of IPv6 connectivity for client/server and peer-to-peer communications.

Depending on programming language and the API, you may or may not need to make changes to the code, but you should perform tests to make sure that your applications operate properly on IPv6-enabled networks.

- **WinSock 2.0 and later**

Follow instructions in the IPv6 Porting Guide. In particular, avoid using Sockaddr or Sockaddr_in. Use SOCKADDR_STORAGE instead. Instead of **GetHostByName**, use **GetAddrInfo**, which can resolve IP and IPv6 addresses. Do not use DWORD or other 32-bit variables for passing addresses; instead, use SOCKADDR_STORAGE.

- **RPC, WinInet, DirectPlay**

In most cases, no code changes are required for programs using these programming interfaces. Avoid passing IP addresses as part of the network protocol; instead, use DNS names if appropriate.

- **User Interface**

In most circumstances, UIs should display DNS names instead of IP or IPv6 addresses. For management-type applications that require the user to view and manipulate IP/IPv6 addresses directly, use controls that can accommodate multiple address formats. Do not parse IP/IPv6 addresses manually; instead, use **GetAddrInfo** or **WSAStringToAddress**.

General Guidelines

Your application is expected to operate equally well over TCP/IP or IPv6, whichever is available. Typically, a well-written application will not have separate code sections specific to IP and IPv6, because the Platform SDK now provides uniform functions and structures for manipulating both types of addresses. These structures and functions are platform-independent, and the same binary is capable of running on any 32-bit Windows platform.

Implementation Details – S4.7

For more information, consult the *IPv6 Porting Guide* available at <http://www.microsoft.com/ipv6>.

F1.8 Networking applications and protocol support

Networking applications should perform their networking functions with the use of TCP/IP and IPv6.

New applications should not make use of IPX/SPX, NetBEUI, AppleTalk, DLC, OSI, or other obsolete networking protocols and APIs. Support for these protocols has either been removed from

Windows XP or will be removed from future versions of Windows. Instead, migrate your application to TCP/IP and IPv6.

Applications should not attempt to replace the TCP/IP stack provided in Windows XP.

Implementation Details – S4.8

To ensure that your application operates properly in legacy-free networking environments, follow these steps:

- Test your applications in TCP/IP-only environment, with no other protocols installed.
- Test with IPv6 installed, to ensure IPv6 compatibility.
- Do not use the **NetBIOS** system call.
- WinSock programmers should not use address families other than AF_INET and AF_INET6.
- Review IPv6 compatibility section in the "Future Requirements" in this document.

F1.9 Create isolated applications using side-by-side assemblies

Creating an isolated application provides many benefits, especially in planning for future changes to the application and Operating System.

- Application Isolation increases reliability by reducing the impact of changes in the system. New versions of a shared assembly won't randomly affect an isolated application. The installation, removal, or upgrading of other applications on the system will not affect an isolated application.
- Applications are more location independent, especially in freeing themselves from COM registration in the registry. A fully isolated application may be installed by using an **XCOPY** command.
- Using versioned side by side components reduces the likelihood of needing a reboot after an update.
- Isolated applications are not tied to the shipping schedule of their side-by-side assemblies. Applications and administrators can update the configuration after deployment without having to reinstall the application.

Isolated applications are built up of safely shared or private side by side assemblies. The application supplies a manifest that describes the application's dependencies. Applications should use side by side components whenever they are available. For example, applications that use the Windows Common Controls, GDI+, MFC, ATL or the C runtime libraries should use a manifest to specify the exact version of components required and isolate the application from any new versions of the components that could potentially harm the application.

Components, and especially COM components, that are used by a single application should be written as private side by side assemblies, described with a component manifest and installed in the applications folder. This is similar to creating private DLLs today, but allows for later safe servicing of the application using publisher configuration policy.

Vendors building components that are shared across applications should develop side by side assemblies, described with a manifest, signed with the publisher’s private key, and installed into the system managed assembly store using the Windows Installer. Developers who distribute these components for inclusion with other applications must provide a Windows Installer Merge Module. Applications should follow the instructions in the Windows SDK, “Installing Side-by-side Assemblies as Shared Assemblies”.

Once distributed, side by side assemblies must never be updated or overwritten. New versions of the assemblies are always installed side by side with older versions on the same system. Developers can use configuration policy files to redirect an application’s use of an older assembly to a newer one. Please see the Windows SDK, “Publisher Configuration” for more information.

Implementation Details – F1.9

The following is an example of an application manifest for an application named MySampleApp.exe. The application consumes the Common Controls side-by-side assembly (COMCTL32).

Sample Manifest - MySampleApp.exe.manifest

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  <assemblyIdentity type="win32"
    name=" MyOrganization.MyDivision.MySampleApp"
    version="6.0.0.0"
    processorArchitecture="x86"
    />
  <dependency>
    <dependentAssembly>
      <assemblyIdentity type="win32"
        name="Microsoft.Windows.Common-Controls"
        version="6.0.0.0"
        processorArchitecture="x86"
        publicKeyToken="6595b64144ccf1df"
        language="*"
        />
    </dependentAssembly>
  </dependency>
</assembly>
```

For more information about isolated applications please see the Windows SDK, “Isolated Applications and Side-by-side Assemblies”

F1.10 Do not use NULL DACLs

Windows objects such as: Files or directories on an NTFS file system, Named pipes, Anonymous pipes, Processes, Threads, etc.) have a Security Descriptor (SD). Windows objects with a Security Descriptor that contains a NULL discretionary access-control list (DACL) have no access restrictions and can create a serious security risk. Your application must not depend on Windows objects having NULL-DACLs, and must not create Windows objects with NULL-DACLs.

This is quite different from using a NULL Security Descriptor (SD). For example, if the SECURITY_ATTRIBUTES structure contains a NULL SD, the object is assigned the default security descriptor of the calling process. A Windows object with a security descriptor which contains a NULL DACL does not inherit access control settings, but instead has no access control restrictions and the object can be modified at will by any other process or user.

For more information, search on “DACLs and ACEs” in the MSDN:
<http://msdn.microsoft.com/library/default.asp>.

F1.11 Do not run as LocalSystem

Whenever possible, services should not run as LocalSystem. Services that run in the context of LocalSystem have broad access rights that have been exploited in attacks by hackers. Use LocalService or NetworkService instead:

- The NetworkService account is a predefined local account. It has minimum privileges on the local computer and acts as the computer on the network. The name of the account is NT AUTHORITY\NetworkService. A service that runs in the context of the NetworkService account presents the computer's credentials to remote servers.
- The LocalService account is a predefined local account. It has minimum privileges on the local computer and presents anonymous credentials on the network. The name of the account is NT AUTHORITY\LocalService.

F1.12 Use Appropriate Sensitive Data Storage

When storing sensitive data for users or system components, a protected store should be used that provides the following:

- An easy-to-use interface that takes data and optional password or other entropy and receives shrouded data.
- Data is protected from other users who are able to log on to the same device.
- Data is protected from tampering while the device is offline.
- Extensibility that allows the use of hardware tokens such as smart cards or biometric devices.

An easy and effective way to meet this requirement and protect sensitive user data is to use the data protection APIs (DPAPI): **CryptProtectData()** and **CryptUnprotectData()**.

An easy and effective way to meet this requirement and store sensitive system component or service data is to use the LSA secrets APIs: **LsaStorePrivateData()** and **LsaRetrievePrivateData()**.

Glossary

Assembly – Fundamental unit for naming, binding, versioning, deploying, or configuring a block of programming code. These code assemblies may be placed in DLLs or COM assemblies. Applications with common functionality may run shared blocks of programming code which are referred to as modules or code assemblies.

Windows XP has an infrastructure for the safe sharing of assemblies, referred to as side-by-side assembly sharing.

Application Manifest – File describing an isolated application. It specifies the information required to run the application, including dependencies on private assemblies, specific versions of shared assemblies and metadata for private assemblies. The name of an application manifest file is the name of the application executable followed by the extension .manifest. For example, for MySampleApp.exe, the manifest file would be MySampleApp.exe.manifest.

Certification Authority (CA) – An entity entrusted to issue certificates asserting that the recipient individual, computer or organization requesting the certificate fulfills the conditions of an established policy

Crash – In the context of this document means that the application stops responding and/or loses data. A failure within the application that does not cause loss of data, displays information that would allow a typical user to understand what went wrong and how to avoid the problem in the future, and allows the user to continue running the application or close it, is not considered a "crash."

CSIDL – These constants provide a unique system-independent way to identify special folders. Used in conjunction with **SHGetFolderPath** and other APIs.

Degrade gracefully – Does not crash the application or the operating system (GPF or blue screen), and does not lose user data. Also, a dialog box or other visual and audio cue appears informing the user of the issue. For example, when a feature requires that users have access rights that they do not have, users should be informed of this when they attempt to use the feature.

DLL Registration – Some DLLs need to have information in the registry in order for the DLL to be used or function fully. Placing this information into the registry is DLL registration.

High Contrast support – An option set by the user indicating that they require a high degree of contrast to improve screen legibility. Some application features may be exempted, such as when the use of color is intrinsic and indispensable to the goal of the feature.

Isolated Application – Application using side-by-side assemblies, but not sharing its assemblies. An isolated application is always accompanied by an application manifest file. The manifest specifies the versions of the shared side-by-side assemblies that the application binds to at run time and may contain some metadata for private side-by-side assemblies. The manifest contains information traditionally stored in the registry; a fully isolated application may be independent of the registry.

HKCU — Abbreviated form of HKEY_CURRENT_USER

HKLM — Abbreviated form of HKEY_LOCAL_MACHINE

Long file name (LFN) — Any filename that exceeds 8.3 characters in length or contains any character that is not valid in the 8.3 namespace.

Side-by-side sharing — A form of sharing in Windows XP, Windows 2000, Windows Me and Windows 98 Second Edition that enables multiple versions of the same DLL to run at the same time.

Universal naming convention (UNC) — The system for indicating names of servers and computers, such as \\Servername\Sharename. UNC paths can indicate deeper paths to individual files located in a shared network resource, with the additional path continuing with additional backslashes. For example:
\\Servername\Sharename\Folder1\Folder2\Filename.txt.

User profile — A computer-based record maintained for an authorized user of a multi-user computer system. A user profile is needed for security and other reasons; it can contain such information as the user's access restrictions, mailbox location, type of terminal, and so on.

Visual Style — User selectable visual configurations provided in Windows XP that include color scheme, fonts, font sizes, and wallpaper settings, as well as the way controls, window borders, and menus are drawn

Windows Installer service — Provides end users with a way to install and remove applications, or components of software as needed. System administrators can more easily manage applications and support roaming users.